$10 Volume XVII, Number 5 January 1996 February

FORTH DIMENSIONS

FORML '95 Report

Stepper Motors

Scattering Colon Definitions

Embedded Systems Conference

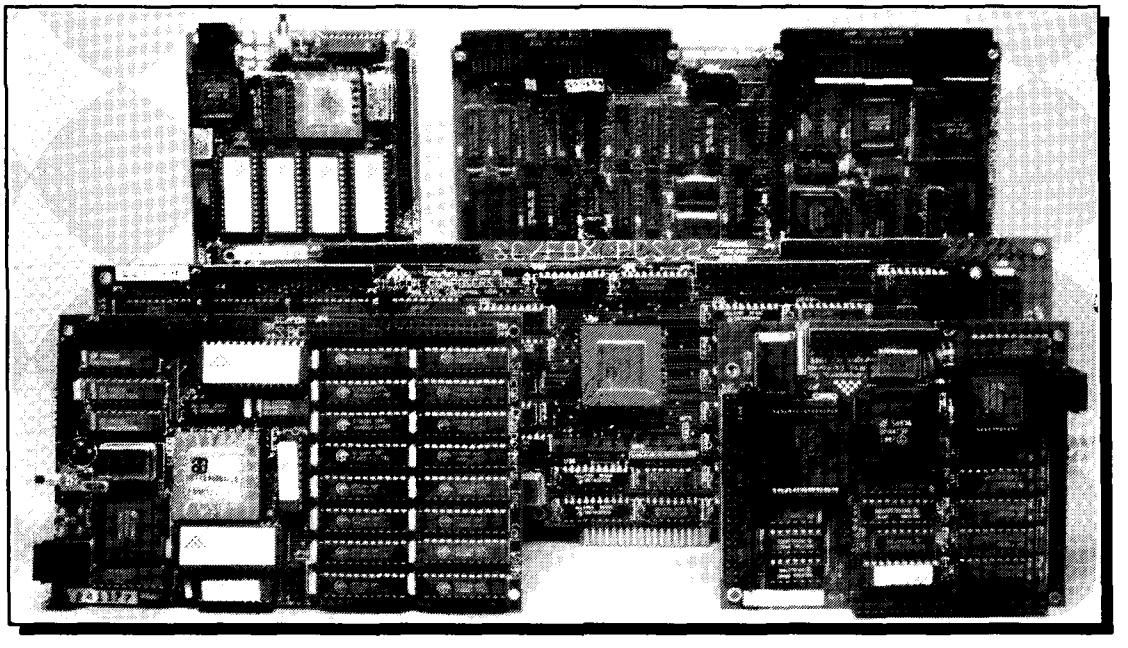
Mobile Computing in Brazil

SILICON COMPOSERS INC

FAST Forth Native-Language Embedded Computers

DUP

>R



C@

R>

Harris RTX 2000^{tm} 16-bit Forth Chip
- 8 or 10 MHz operation and **15 MIPS** speed.
- 1-cycle 16 x 16 = 32-bit multiply.
- 1-cycle 14-prioritized interrupts.
- two 256-word stack memories.
- 8-channel I/O bus & 3 timer/counters.

SC/FOX PCS (Parallel Coprocessor System)
- RTX 2000 industrial PGA CPU; 8 & 10 MHz.
- System speed options: 8 or 10 MHz.
- 32 KB to 1 MB 0-wait-state static RAM.
- Full-length PC/XT/AT plug-in (6-layer) board.

SC/FOX VME SBC (Single Board Computer)
- RTX 2000 industrial PGA CPU; 8, 10, 12 MHz.
- Bus Master, System Controller, or Bus Slave.
- Up to 640 KB 0-wait-state static RAM.
- 233mm x 160mm 6U size (6-layer) board.

SC/FOX CUB (Single Board Computer)
- RTX 2000 PLCC or 2001A PLCC chip.
- System speed options: 8, 10, or 12 MHz.
- 32 KB to 256 KB 0-wait-state SRAM.
- 100mm x 100mm size (4-layer) board.

SC32^{tm} 32-bit Forth Microprocessor
- 8 or 10 MHz operation and **15 MIPS** speed.
- 1-clock cycle instruction execution.
- Contiguous 16 GB data and 2 GB code space.
- Stack depths limited only by available memory.
- Bus request/bus grant lines with on-chip tristate.

SC/FOX SBC32 (Single Board Computer32)
- 32-bit SC32 industrial grade Forth PGA CPU.
- System speed options: 8 or 10 MHz.
- 32 KB to 512 KB 0-wait-state static RAM.
- 100mm x 160mm Eurocard size (4-layer) board.

SC/FOX PCS32 (Parallel Coprocessor Sys)
- 32-bit SC32 industrial grade Forth PGA CPU.
- System speed options: 8 or 10 MHz.
- 64 KB to 1 MB 0-wait-state static RAM.
- Full-length PC/XT/AT plug-in (6-layer) board.

SC/FOX SBC (Single Board Computer)
- RTX 2000 industrial grade PGA CPU.
- System speed options: 8, 10, or 12 MHz.
- 32 KB to 512 KB 0-wait-state static RAM.
- 100mm x 160mm Eurocard size (4-layer) board.

For additional product information and OEM pricing, please contact us at:
SILICON COMPOSERS INC 655 W. Evelyn Ave. #7, Mountain View, CA 94041 (415) 961-8778

Contents

Features

5 **Scattering a Colon Definition** — *M.L. Gassanenko*

The author's prefix-notation Forth assembler performs initialization actions before processing any new instruction. The assembler's switches are set according to defaults and instruction operands, and they determine what happens; some switches select a Forth word to be executed. The problem was that initialization actions belong to two different modules at the same time: to the module they initialize and to the general initialization module. The author wanted to distribute these actions so they would be located in the modules they initialize, but used as a single definition.

8 **Mobile Computing in Brazil** — *Klaus Blass*

Brazil boasts the world's eighth-largest economy and imports now enter freely, including computer hardware. Local companies are investing in sales force automation, setting the stage for mobile computing solutions such as those this author's company develops in Forth. While other firms are developing for the same market, Forth makes maximum use of limited resources, typically packing four times the functionality of a competitor's C program into an executable one-half to one-fourth the size.

18 **FORML 1995** — *András Zsótér*

Once again, intrepid Forth practitioners converged on Pacific Grove, California, to present their latest works, to collaborate informally on new ideas, and to seek a concensus about technical, organizational, and political issues facing the Forth community. Our reporter from Hong Kong reports on his first experience of the annual FORML Conference.

19 **Stepper Motors** — *Skip Carter*

This is the first installment of the new "Forthware" column about using Forth to interact with the real world. It will explore how to control motors of various types; this issue discusses using the PC parallel port to control stepper motors—adopting the useful fantasy of working on a microprocessor-based control application and using the PC parallel port as a proxy for the digital I/O channels on the controller.

Departments

4 **Editorial** ..... Forth jobs, FIG works.

4 **Dot-quote** ..... Postpone, the inevitable.

7 **Advertisers Index**

10 **FIG Board Meeting** ... "New faces, fresh approaches."

11 **President's Letter** ... Skip Carter takes the helm.

12 **Embedded Systems Conference** & **"A Prayer for Forth"**

13 **Stretching Forth** ..... All the standard Forth you need.

42 **Fast Forthward** ..... The prospects for ++Forth.

Editorial

Forth Dimensions
Volume XVII, Number 5
January 1996 February

Published by the
Forth Interest Group

Editor
Marlin Ouverson

Circulation/Order Desk
Frank Hall

We were very happy when Everett (Skip) Carter agreed to write a column for *Forth Dimensions*, because we knew that his background brought him very well qualified to the task. The first installment appears in this issue and tackles stepper motors, a classic subject to which we long have wanted to do justice. Elected to the FIG Board of Directors, and then elected by the Board to be its new President, Skip's first official communication as such is also in this issue. We congratulate him and thank him in advance, as we thank past-president (and still Board member) John Hall for his years of service.

In the preceding issue of *FD,* John outlined his views of the tasks facing the new Board and asked us to give it our support and encouragement. The Board's first meeting was propitious, even in the face of some difficult decisions, with each attending member demonstrating the desire to make FIG a more vital and solid organization, increasingly of service to its members and in a better position to promote Forth. We wish the entire Board success and welcome the contributions of each of its members.

Forth expertise pays—In her report from the November meeting of FIG's new Board of Directors, Elizabeth Rather mentions, "The programmer job referral service recently implemented by Frank Hall in the FIG office will be formalized and expanded." At the time of this writing, the office knows at least one company looking for Forth programmers to work on Open Firmware projects for the PowerPC, with responsibility for all phases of software development and, possibly, for debugging new silicon and motherboards.

This is just one example of the boost given by the IEEE Open Firmware standard to those with Forth expertise. It and ANS Forth soon may prove to have been instrumental in reinvigorating market recognition of and appetite for Forth. Promising signs already have been reported by some vendors and consultants, and we likely can expect the trend to continue. For the career-minded, this is an excellent time to stay abreast of technical developments, to keep one's skills up to date, and to enjoy FIG's growing ability to provide job referrals to its members.

—Marlin Ouverson
editor@forth.org

dot-quote

Once I learned Forth, I thought [COMPILE] and COMPILE made things quite clear. I could tell when something was to happen for these two similar words. I'm still trying to grasp POSTPONE.

—Dwight Elvey, elvey@hal.com

See whether this fits—you know what COMPILE does to a word that isn't immediate, and you know what [COMPILE] does to a word that is immediate.

POSTPONE acts like [COMPILE] on the immediate words and like COMPILE on the non-immediate ones. So you don't have to know whether a word is immediate or not, you always know which word to use.

—Jonah Thomas, JEThomas@ix.netcom.com

(Quoted from comp.lang.forth with permission.)

Forth Dimensions welcomes editorial material, letters to the editor, and comments from its readers. No responsibility is assumed for accuracy of submissions.

Subscription to *Forth Dimensions* is included with membership in the Forth Interest Group at $40 per year ($52 overseas air). For membership, change of address, and to submit items for publication, the address is: Forth Interest Group, P.O. Box 2154, Oakland, California 94621. Administrative offices: 510-89-FORTH. Fax: 510-535-1295. Advertising sales: 805-946-2272.

Copyright © 1996 by Forth Interest Group, Inc. The material contained in this periodical (but not the code) is copyrighted by the individual authors of the articles and by Forth Interest Group, Inc., respectively. Any reproduction or use of this periodical as it is compiled or the articles, except reproductions for non-commercial purposes, without the written permission of Forth Interest Group, Inc. is a violation of the Copyright Laws. Any code bearing a copyright notice, however, can be used only with permission of the copyright holder.

The Forth Interest Group

The Forth Interest Group is the association of programmers, managers, and engineers who create practical, Forth-based solutions to real-world needs. Many research hardware and software designs that will advance the general state of the art. FIG provides a climate of intellectual exchange and benefits intended to assist each of its members. Publications, conferences, seminars, telecommunications, and area chapter meetings are among its activities.

"*Forth Dimensions* (ISSN 0884-0822) is published bimonthly for $40/46/52 per year by the Forth Interest Group, 4800 Allendale Ave., Oakland, CA 94619. Second-class postage paid at Oakland, CA. POSTMASTER: Send address changes to *Forth Dimensions*, P.O. Box 2154, Oakland, CA 94621-0054."

Scattering a Colon Definition

M.L. Gassanenko
St. Petersburg, Russia

The Problem

The technique presented here is used in a prefix notation Forth assembler that performs initialization actions (variables resetting, etc.) every time before processing a new instruction. The assembler has many switches that get set according to the defaults and the instruction operands and then determine what to do. Some switches are "functional": executing them executes a Forth word, setting or resetting determines which word is to be executed. (The switches are implemented as multi-CFA words.) Due to these numerous switches definitions of instruction groups are as readable as instruction formats, but the initialization code grows. Had the initialization code been written as a separate definition, it would occupy two dense-typed block screens; sequential files could not be a solution because one dense screen of sequential file isn't better than two blocks.

So, the problem is that initialization actions belong to two different modules at the same time: to the module they initialize and to the general initialization module. We want to distribute these actions so they will be located in the modules they initialize, but used as a single definition.

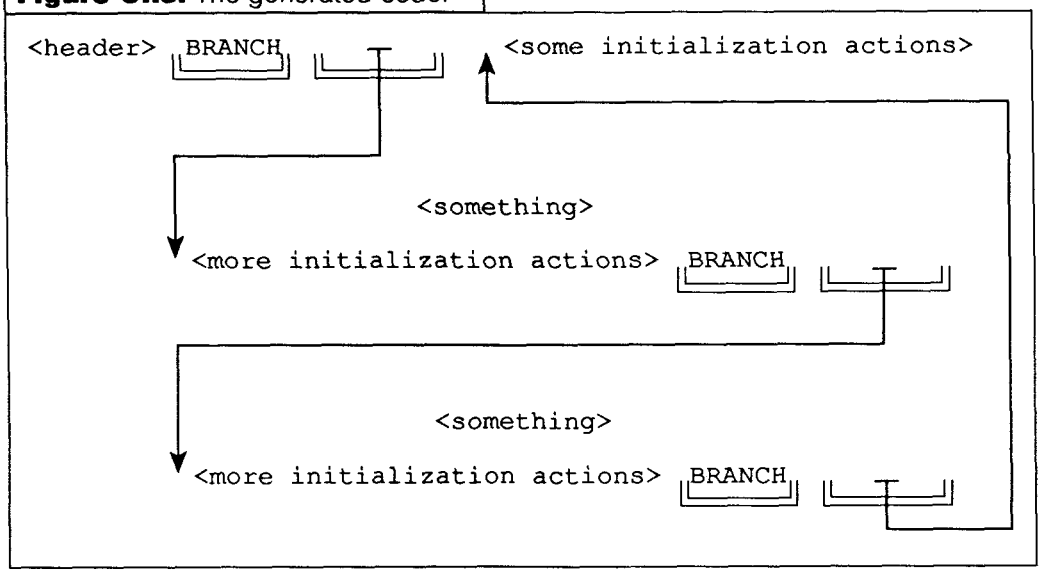
The Solution

The solution scatters the code of the initialization definition over the screens where the things to be initialized are used.

The words `...` , `..:` , and `;..` are used as follows:

```
: INIT ... <some initialization actions> ;
<something>
..: INIT <more initialization actions> ;..
<something>
..: INIT <more initialization actions> ;..
<and so on>
```

Figure One. The generated code.



The generated code looks like Figure One.

Implementation

The implementation code below is non-standard, but very portable.

With the definitions:

```
\ Fetch/store a reference that, e.g.,
\ follows a BRANCH.
\ The branch addresses are relative.
: REF@ ( orig -- dest ) DUP @ + ;
: REF! ( dest orig -- ) TUCK - SWAP ! ;

\ Add size of a compiled token.
: TOKEN+ ( addr -- addr' ) CELL+ ;
```

We might define:

```
\ Forth-83 Standard words that build
\ references after BRANCHes.
```

```
: >MARK     ( -- orig ) HERE 0 , ;
: >RESOLVE  ( orig -- ) HERE SWAP REF! ;
: <MARK     ( -- dest ) HERE ;
: <RESOLVE  ( dest -- ) HERE CELL ALLOT
                        REF! ;
```

And now we can define:

```
: ... COMPILE BRANCH >MARK >RESOLVE
  ; IMMEDIATE
: ..: ' >BODY TOKEN+ DUP REF@ SWAP
  >RESOLVE !CSP 400 ] ;
: ;.. 400 ?PAIRS ?CSP COMPILE BRANCH
  <RESOLVE [COMPILE] [ ; IMMEDIATE
```

The word ... should be the first word in a definition. It compiles a BRANCH to code that follows it, which is equivalent to no-operation until you modify the definition by means of ..: and ;... The words ..: (fetches a definition name from input stream and starts compilation) and ;.. (finishes compilation) are similar to : and ; with the difference that they do not redefine the name, but add the compiled code to the definition. This implementation does not check if the definition starts with a ... and, if you omit ... your system is likely to hang.

In F-PC there may be some problems with long jumps and long addresses. Note that a new branching word is defined:

```
: BRANCHL 2R> REF@L 2>R ;
```

because the F-PC BRANCH cannot cross the segment boundaries. F-PC with its double-cell addresses isn't well-suited for return address manipulations and code generation tricks. The F-PC code is given in Listing One.

Why a Special Construct

The evident benefit of this tool is that the programmer does not have to modify the initialization definition when he adds a new mechanism to the growing program. Deleting a mechanism also becomes painless: if you do not load a block, its initialization does not get compiled.

In F-PC this problem is usually solved by means of DEFER variables. We think a special construct is better because it is:

1. laconic;
2. more readable: the purpose may be understood at the first glance;
3. uses no auxiliary names (which have no meaning in themselves).

Conclusion

The technique presented here enables the programmer to distribute fragments of code that should execute as one definition across the modules they logically belong to.

M.L. Gassanenko began his serious use of Forth at Leningrad University. In 1989, he implemented backtracking via return stack manipulations, e.g.:

```
: enter >r ;
: 1-10  ( --> i ) ( <-- i )
        1
        begin   r@ enter ( success )
                1+
        dup 10 > until
        drop
        rdrop exit ( failure )
;
: print1-10 1-10 dup . ;
```

...which he calls, "probably my most felicitous work."

The author has created two Forth systems, one for Soviet IBM/370 compatibles; the other is a 32-bit Forth for the 8086/88/188 with a trick that allows both linear and segment:offset forms (reported at euroForth '94). His most recent work is a formalization of control transfers due to return stack manipulations (presented at euroForth '95).

He graduated from St. Petersburg University's Department of Applied Mathematics and Control Processes in 1992, and is now a post-graduate student at SPIIRAN (St. Petersburg Institute of Informatics and Automatization, at the Russian Academy of Sciences). He can be reached at gml@ag.pu.ru or at gml@agspbu.spb.su via e-mail.

Listing One. The F-PC code.

```
anew scatter.seq

: REF@L ( orig-seg orig-off --- dest-seg dest-off )
          2DUP   2+ @L XSEG @ +   -ROT   @L ;
: REF!L ( dest-seg dest-off orig-seg orig-off --- )
          2DUP 2>R   !L    XSEG @ - 2R> 2+ !L ;
: TOKEN+ 2+ ;

: >MARKL    ( -- Dorig ) XHERE 0 0 X, X, ;
: >RESOLVEL ( Dorig -- ) XHERE 2SWAP REF!L ;
: <MARKL    ( -- Ddest ) XHERE ;
: <RESOLVEL ( Ddest -- ) XHERE 0 0 X, X, REF!L ;

: BRANCHL 2R> REF@L 2>R ;
: >TCODE ( cfa -- seg off ) >BODY @ XSEG @ + 0 ;

: ?PAIRS XOR ABORT" NON-PAIRED WORD" ;

: ... COMPILE BRANCHL >MARKL >RESOLVEL ; IMMEDIATE
: ..: ' >TCODE TOKEN+ 2DUP REF@L 2SWAP >RESOLVEL !CSP 400 ] ;
: ;.. 400 ?PAIRS ?CSP COMPILE BRANCHL <RESOLVEL [COMPILE] [ ; IMMEDIATE
```

ADVERTISERS INDEX

The Computer Journal 7

FORTH, Inc. 7

Forth Interest Group .... centerfold

Laboratory Microsystems, Inc. (LMI) 7

Miller Microcomputer Services 23

Rochester Conference 1996 44

Silicon Composers 2

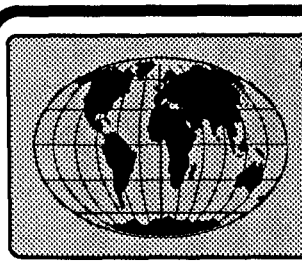
Total control with *LMI FORTH*™

For Programming Professionals: an expanding family of compatible, high-performance, compilers for microcomputers

For Development:
Interactive Forth-83 Interpreter/Compilers for MS-DOS, 80386 32-bit protected mode, and Microsoft Windows™
- Editor and assembler included
- Uses standard operating system files
- 500 page manual written in plain English
- Support for graphics, floating point, native code generation

For Applications: Forth-83 Metacompiler
- Unique table-driven multi-pass Forth compiler
- Compiles compact ROMable or disk-based applications
- Excellent error handling
- Produces headerless code, compiles from intermediate states, and performs conditional compilation
- Cross-compiles to 8080, Z-80, 64180, 680X0 family, 80X86 family, 80X96/97 family, 8051/31 family, 6303, 6809, 68HC11
- No license fee or royalty for compiled applications

Laboratory Microsystems Incorporated
Post Office Box 10430, Marina Del Rey, CA 90295
Phone Credit Card Orders to: (310) 306-7412
Fax: (310) 301-0761

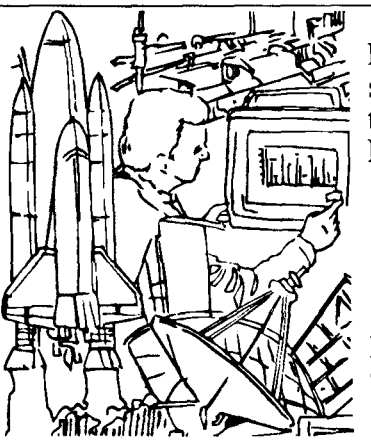


The Computer Journal

Support for older systems
Hands-on hardware and software
Computing on the Small Scale
Since 1983

Subscriptions
1 year $24 - 2 years $44
All Back Issues available.

TCJ
The Computer Journal
P.O. Box 3900
Citrus Heights, CA 95611-3900
800-424-8825 / 916-722-4970
Fax: 916-722-7480
BBS: 916-722-5799



From NASA space systems to package tracking for Federal Express...

chipFORTH

...gives you maximum performance, total control for embedded applications!

- Total control of target kernel size and content.
- Royalty-free multitasking kernels and libraries.
- Fully configurable for custom hardware.
- Compiles and downloads entire program in seconds.
- Includes all target source, extensive documentation.
- Full 32-bit protected mode host supports interactive development from any 386 or better PC.
- Versions for 8051, 80186/88, 80196, 68HC11, 68HC16, 68332, TMS320C31 and more!

Go with the systems the pros use... Call us today!

FORTH, Inc.
111 N. Sepulveda Blvd, #300
Manhattan Beach, CA 90266
800-55-FORTH 310-372-8493
FAX 310-318-7130 forthsales@forth.com

Forth's competitive edge:

Mobile Computing in Brazil

Klaus Blass
Rio de Janeiro, Brazil

This article has its origin in one of my postings to comp.lang.forth on USENET, where I commented on a discussion about the "Forth scare." The *FD* editor then invited me to write about my experience with successfully selling the Forth approach. Of course, the situation in Brazil may not be directly comparable to that in the U.S. or Europe (it may well be to the new Eastern European democracies and Russia, though). But I think some aspects are common everywhere.

Some time has passed now, and I never found the time, but when I saw an article about "Forth in the HP100LX" in the November/December 1995 edition of *FD*, I decided that now was the time to write about my experience, including about some of the things that Forth can do on the HP100LX.

Brazil is the size of the United States without Alaska, and has a population of 160 million people. It boasts the world's eighth-largest economy and only a few years ago reverted from a military dictatorship to a democratic government. It has since started to open its economy by removing trade barriers and heavy import duties. During the dictatorship, a "reserved market" for computing technology had been introduced, meaning imports were made almost impossible in order to protect the national computing industry. The result was disastrous. The local computer manufacturers cloned outdated technology and sold it at outrageous prices. This has now changed as imports are entering freely and the economy is trying to catch up on two lost decades.

What we sell to the customer is not a computer language but a complete solution.

In the process of streamlining their operations, a lot of companies have started to invest in sales force automation. This sets the stage for mobile computing solutions such as those my company develops.

Many of our projects involve HP palmtops as a computing platform. These very neat little XT-class computers fit into a (large) shirt pocket and typically run for weeks on two AA batteries. Their main limitations for doing sophisticated work are speed (8 MHz 80186) and memory (one or two Mb shared between system memory and file storage). While there are other companies developing programs for these palmtops, typically in C or even Cobol, their executables are little elephants around 500 Kb, leaving just enough memory for them to execute.

In comes Forth. What better language can be imagined to make maximum use of limited resources? We typically pack something like four times the functionality of a competitor's C program into an executable one-half to one-fourth the size (typically 50 to 80 Kb). Recently, a plastics manufacturer looking for a portable sales automation solution approached us and told us all other software developers had told them what they want to do needs a four Mb notebook computer two to five times the price of the palmtop, not to mention the weight. Judging from similar projects we have done, I estimate the size of the executable to be about 70 Kb, leaving ample room for files on a two-Mb palmtop. Now this customer plans to equip 150 salespersons with the system, initially. With a price difference of at least $1000 between a palmtop and a notebook, which software developer do you think they chose? We have done a similar project, for one of Brazil's two large breweries, which eventually will involve several thousand machines!

Obviously, one of the first things a prospective customer asks us is: How do you do it? And we tell him: We are doing it in Forth! He may or may not have heard of Forth before (actually a surprising number of people here have heard about Forth!), he usually raises questions about who will be able to support the system, just in case our company disappears from the scene, etc. In short, he raises all those arguments in favor of a mainstream development environment but, in the end, a factor of two to five in hardware costs is convincing, especially if you are talking about equipping hundreds or thousands of salespersons with hardware.

When I was first contacted by the company I am working with now, they had a programming job for me and asked, how much and how many months? I told them one month if I could do it in Forth, two to three months

if I had to use C. They had never heard about Forth before, so I told them. And I showed them a comparison between the "Hello world" sample program included with the C compiler and a (metacompiled) "Hello Forth world" example that was about 1/25th the size. They accepted that I did the project in Forth and, after completing it, asked me to join the company. Of course, they had realised the potential of Forth for their palmtop solutions, and now, two years later, they have no reason to regret their decision.

When our sales people approach a new customer, there comes the moment where they have to mention Forth and they copy my initial approach with them. "If you insist we do it in C, it will take so much longer and the program will be so much bigger." Next thing, I do a well-prepared presentation on Forth to the customer, so he sees why Forth yields these results. The third step is to convince the customer of guaranteed support for his product, and this is something I think we have to do. Whereas my president, by now, is a bit anxious about not giving away too much of our "secret weapon," I think we have to invest in popularising Forth, training more programmers, hiring university students as trainees, etc. We should always be a step ahead of the others, but I think competing software developers using Forth will give us even more credibility. I must mention, however, that so far not a single customer has backed off because of Forth. What we sell the customer is not a computer language but a complete solution. Let him compare this solution to others and then decide whether a non-mainstream development environment is acceptable or not. In other words, don't sell Forth, sell the results you can obtain with it! Interestingly enough, after having done this for two years now, prospective customers already have another image of Forth. Instead of saying "I hope you use C," they say, "We have heard you use this crazy magic language which allows you to do things no one else can." We are also getting the reputation that our programs are faster than those of the competitors, which in general they are not. But I can always optimise code at critical points and show them the improvement I got.

By the way, I think a similar line of reasoning holds for another discussion on comp.lang.forth about Chuck Moore's P21 Forth-like microprocessor. I received my evaluation kit some time ago and what I would like to do is to use it in some prototype application and show this to the right people. (I have got some ideas where the P21's integrated NTSC controller, the fact that it doesn't need a lot of supporting chips and, thus, the low price of a resulting device will come in very handy.) I won't try to sell them the Forth-chip idea, I'll try to sell them a product they didn't think was possible for this price. When they want to know how I do it, then I'll tell them about Forth chips!

Encouraged by Edward Borasky's article in a recent issue of *FD* (XVII/4, *op. cit.*) that shows we are not the only ones using Forth on the HP palmtops, I would like to talk a little more about this.

Packing more functionality than non-Forth competitors into relatively inexpensive hardware is not the only advantage we have. Another well-known feature of Forth is its ease of accessing non-standard hardware. The HP palmtops have a number of non-standard hardware features. One example is a three-stage zoom, sometimes very useful because of the tiny LCD screen. None of the competition's systems I have seen make use of this feature in their programs (although you could, of course, call the appropriate interrupt from a C program as well). In Forth, it takes me a minute to define three words—`ZOOM1`, `ZOOM2`, and `ZOOM3`—and I have extended the language once and for all to include this feature. Similarly, we make optimal use of the LCD's non-standard "colors," four different shades of grey. For example, many of our screens have scrollable windows representing tables of product or customer information. The readability and appeal of such tables increases drastically by changing the white background of every second line to a light grey shade (think of that computer-printer stationery that shows every second line in a different background color). The visual effect is stunning. And the guys who buy your solution are very susceptible to aesthetic impressions. Even if another software house's solution is just as good as yours, but your application looks much more professional and appealing, the customer will most certainly go with your solution.

Another example of non-standard hardware is HP's proprietary infrared interface on the HP100LX. Despite minimal documentation, it took me a weekend to develop a driver for printing on a portable HP infrared thermal printer they had developed for some scientific calculator. This printer is not much bigger than the palmtop itself and can be very handy for a salesman to print a copy of the order he has just taken and leave it with the customer.

Other features easily implemented in Forth are words to check and display the palmtop's main and backup battery status, and a word to control the screen contrast from within the program. After running the little demo program on the HP200LX, which displays some text in a much larger font, I started to define my own "bigfont." The letters are 32 x 48 dots, which permits four lines of 20 characters. The implementation uses graphics mode, writes 32-bit words directly to the video memory, and has a few words to emit characters, display strings, and control cursor position. Nothing very fancy, but you can read this font from a distance of five meters without effort. I plan to use it in situations like price surveys in a supermarket, where you would put the palmtop somewhere on the shelf in front of you and then start reading in items with a connected bar-code reader. In any case, no one else has it, and I can demonstrate it to prospective customers, be it only to show the name of their company on an opening screen.

I hope that this little article is able to contribute something to the morale of the Forth community, mainly because I am not stating theoretical advantages, but relating some real facts where Forth is helping us to get ahead of the competition.

Klaus Blass got his B.S. in mathematics and computer science from the University of Cape Town. He worked as a systems engineer with IBM in South Africa, as a consultant and project manager with PRIME Computer in Kuwait and Saudi Arabia, and is now technical director of a mobile computing company in Rio de Janeiro, Brazil. When forced to work with computers, he preferably uses Forth; when not forced to work at all, he prefers to go on an expedition in the Amazon jungle. He can be reached at klausb@ax.apc.org.

A Report of the Board Meeting Nov. 27–28, 1995

New Faces, Fresh Approaches in New FIG Board

Elizabeth D. Rather
Manhattan Beach, California

The new Forth Interest Group (FIG) Board of Directors elected by the membership last spring has elected a new slate of officers and adopted new policies for 1996.

Skip Carter, elected to the board by an overwhelming 92% of voting members, will become the new President. Jeff Fox will be Vice President, Mike Elola will continue as Secretary, and Andrew McKewan will assume accounting responsibilities as Treasurer.

Major policy changes include a revised membership program, opening of advertising privileges to companies with products that may not be directly Forth-related, adjustments in shipping charges for books and software, expanded programming job referrals, and substantially improved electronic services.

The Board considers its top priority to be ending the deficit financing of the past few years by increasing revenues and cutting expenses, in order to continue to provide a high level of member services. To this end, the Board evaluated the three major activities of FIG: publication and distribution of *Forth Dimensions*, sales of books and software, and the annual FORML conference. In each case, strategies were developed to increase revenues and reduce expenses without significant loss of service. As a result, the 1996 budget adopted by the Board includes a significant allocation for "outreach" (actions designed to increase awareness of Forth in general and membership in particular) and a small surplus to cover contingencies.

The most immediate impact on members will be an increase in dues, effective March 1, 1996, from $40 per year to $45. This modest increase is the first in several years. Renewals and new subscriptions received prior to March 1 will be at the previous rate. International mailing charges were also increased to $15 to cover actual costs. Student rates remain unchanged.

New membership categories were also established: Corporate memberships, at $125 per year, include five copies of each issue of *Forth Dimensions* and other benefits; Library memberships, also at $125, include a complete set of issues for the year sent in a single package at the end of the year plus a copy of the FORML Proceedings, in addition to regular bi-monthly issues.

In the product arena, an analysis of shipping and handling costs led to a revised shipping fee schedule which will be effective immediately (see the price list in this issue). Other pricing issues will be evaluated over the course of the year.

The overall budget for production of *Forth Dimensions* and general services will be reduced significantly, by allocating more responsibility to volunteers, finding lower-cost alternatives for some services, and in some cases reducing hours. Although it is hoped that less expensive production alternatives for *Forth Dimensions* may be found, it was pointed out that the major costs for this are in the labor for editing, formatting and layout, production, and related activities.

The Board considered electronic publication of *Forth Dimensions* as a possible cost-saving alternative. However, unless paper production is discontinued altogether, production of a separate electronic edition actually increases editorial costs significantly, for a negligible reduction in postage and printing. A recent survey of FIG members indicates that about 75% do not yet use on-line services regularly.

The Board also voted to extend advertising privileges in *Forth Dimensions* to all firms with products of interest to FIG members. The previous policy limited advertising to Forth-related products. The new policy is expected to offer significantly increased advertising revenues.

The programmer job referral service recently implemented by Frank Hall in the FIG office will be formalized and expanded. Any member in good standing may list his availability on the job market; such a listing will appear in the FIG Worldwide Web page for one month, and will be distributed to all inquiring companies. In addition, the job seeker may send a self-addressed stamped envelope and get a list of all current job openings. Currently, approximately 30 programmer and job listings are registered in the FIG office. In addition, corporate members offering programming services may be listed in *Forth Dimensions*.

As the Board recognizes the critical importance of good communication within the growing segment of the Forth community having access to online services, the Internet services provided by Skip Carter at taygeta.com will be

(Continues on next page.)

A Message from the President of the Forth Interest Group

Letter from the President

Everett (Skip) Carter
Monterey, California

December 3, 1995—It is a great honor for me to be writing this letter to you as the new president of the Forth Interest Group. I extend my thanks to all of you for electing me to the Board of Directors, and to the Board of Directors for choosing me as the president.

As I have stated in the past, FIG is *member supported.* As such, I will rely heavily upon your constructive contributions to help FIG grow and to improve the services it provides. I know that each of you has ideas on what you expect of FIG. And so I am issuing a general "call for volunteers" to lend a hand. If you can make the time to volunteer to support FIG activities, contact us and let us know what type of things you can do. If you want to start a *specific* project that you think FIG should be doing (like reviving a defunct local chapter), let us know.

The new Board of Directors is geographically diverse (but still North American this time around) and will rely heavily upon electronic communications between ourselves. Happily, this provides a means for our geographically diverse membership (globally diverse, we have no off-planet members yet!) to communicate with us. Providing all goes well, by the time you read this in *Forth Dimensions* you will be able to contact FIG directly on the Internet at the following addresses:

board@forth.org	all FIG Board of Directors
editor@forth.org	Marlin Ouverson, *FD* editor
office@forth.org	John Hall, FIG business office
pres@forth.org	Skip Carter, FIG President
sec@forth.org	Mike Elola, FIG Secretary
treas@forth.org	Andrew McKewan, FIG Treasurer
vp@forth.org	Jeff Fox, FIG Vice President

The address board@forth.org is a "broadcast" mailbox: mail sent there is automatically forwarded to *all* members of the Board of Directors (Skip Carter, Jeff Fox, Mike Elola, Andrew McKewan, John Hall, Al Mitchell, Brad Rodriguez, Elizabeth Rather, and Nicholas Solntseff). If you have *Forth Dimensions* subscription questions, these should be directed to office@forth.org, not editor@forth.org.

The FIG Web page will move off of taygeta to http://www.forth.org (actually this *is* still on the system taygeta, which will now service the forth.org domain).

Based on the returns of a recent survey of FIG members, only 25% of you can access the Internet. You can always reach us by regular mail, phone, or fax at the FIG office. If you would like to contact me directly, you can send a fax to my business office at 408-626-3735.

However you do it, talk to us. As I used to say to my students, I can't be assured that I am providing for your needs if I don't get any feedback.

(Continued from previous page.)

expanded. A special "domain" has been set up for FIG, and addresses have been set up for communicating with various officers and departments in FIG; for details, see the President's Letter in this issue. Brad Rodriguez will officially represent FIG on the newsgroup comp.lang.forth. Additional planned enhancements will be announced as they are implemented.

Individual Board members have also assumed responsibility for various aspects of FIG operation, as follows:

FIG Chapter relations	Jeff Fox
Education	Al Mitchell and Nick Solntseff
Electronic services	Brad Rodriguez and Skip Carter
FORML	John Hall (Bob Reiling, manager)
Forth Dimensions	Elizabeth Rather (Marlin Ouverson, editor)
Office Management	John Hall (Frank Hall, office manager)
Outreach	Al Mitchell
Products (books and software)	Mike Elola (Frank Hall, sales and shipping)

Those presently having operational responsibility in these departments, such as Frank Hall in the FIG office and Marlin Ouverson at *Forth Dimensions*, will continue as before under the oversight of the responsible Board member.

The new Board unanimously wishes to extend a vote of thanks to John Hall for his dedicated service as President of FIG for the last several years.

Forth Booth at Embedded Systems Conference

Through the efforts of Doug Hammed, Tom Vail of Mosaic Industries, and a few Silicon Valley FIG Chapter members, FIG was able to receive visitors last September 12–14 at the Embedded Systems Conference in San Jose, California.

Doug had encouraged Tom Vail to get through to the show's producers. Just days before the show, things began to move as far as securing a booth for FIG. Signage was prepared by Tom Vail, who volunteered to become a booth partner. He offered to staff the booth and was in agreement that booth visitors would be treated as FIG/Forth prospects first and foremost.

Various Forth-promotional materials were handed out at the show, including the FIG mail-order form and the "Top Ten Reasons for Choosing Forth." A piece from Forth, Inc., called "Choosing Forth" was distributed and, in fact, Elizabeth Rather (president of Forth, Inc.) flew in on the last day to work the booth.

After the event, everyone agreed that the FIG booth had been in a good location, and most agreed that the overall show continues to grow bigger from year to year. Some thought more concrete (and professional) ties between Forth and the embedded systems field should be brought to the show. This year, as in the past, small Forth-controlled mechanical devices were shown, such as an Etch-a-Sketch that responds to Forth commands.

Doug's training and experience is in high technology sales and marketing. He urges FIG and its chapters to be more concerned with how the Forth community appears to others. For Forth to get more of a reputation as a professional solution language, a public-relations effort may be needed that, at least for the Embedded Systems Conference, emphasizes the power of using Forth in these applications. With better-coordinated collateral, signage, and displays, we can help overcome the credibility gap that is natural for those who are hearing about Forth for the first time or those who have heard that Forth is a write-only language.

Doug acknowledged how booth visitors enjoyed the fun flavor and down-to-earth friendliness—we excel in that regard. The fun and warmth of the FIG volunteers is important, but so is getting Forth into a position of respect and credibility regarding industrial and consumer applications.

The perceived friendliness may stem from the fact that the booth is usually well staffed with volunteers who are eager to make conversation with passers-by. Doug feels that SV-FIG's readiness to train new users of Forth is an untapped asset.

—Reported by Mike Elola
based on a talk by Doug Hammed

As usual, it was good to see the members of FIG and other vendors in the Forth community. It was also good to meet a lot of people from the embedded systems community—many of whom were interested in Forth but weren't sure how it can help them. It was disappointing to meet so many people who were surprised to learn that Forth is still alive. But I think the Forth booth gave people the impression that Forth is here to stay. We also put out the word that Forth is a perfect solution for embedded systems programming and that many vendors—large and small—are taking advantage of Forth-based solutions. Adding the PowerPC (with its Open Boot standard) to the list of products utilizing the power of Forth added momentum to the message. Overall, it was a very good experience for us, it was fun working with FIG again, and I hope we will see everyone at next year's show.

—Tom Vail, Marketing Director of Mosaic Industries

A Prayer for Forth

When I undertook a quixotic crusade
On behalf of the Forth tongue obscure
I offered my book to the publishing trade
Who soon beat a path from my door!

O Forth, thy Forth-right endearing concision
Thy power, thy speed, that my book doth convey
Are these but destined for scorn and derision,
Doomed ne'er to go Forth unto light of day?

O Forth, doth thy star now wane in declension?
Declare it a FALSEhood lest we wax too sad.
Pray, Gresham, thy Law now hold in suspension
That for once the Good may drive Forth the Bad.

(An author who looks not for power nor pelf
Will find what he seeks should he print it himself!)

—Julian V. Noble, 1992

Stretching Forth

All the Standard Forth You Need

Wil Baden
Costa Mesa, California

This completes the first year of Stretching Forth. During that time I have stuck close to Standard Forth. The year has shown that Standard Forth is very good, except for file handling. File handling in Standard Forth for anything but the simplest task is a pain, and will be discussed in another article.

In this article I review the Standard Forth words as used in Stretching Forth. They are fundamentally the words I have worked with since long before the Technical Committee first started to meet.

In anything I write for Stretching Forth, I will give definitions or explanations for words (other than NOT) that are not Standard Forth.

Working Words

The most important words in Standard Forth are those given in the Core wordset, section 6.1. In a Standard implementation all of these words are required. There are some of them that I don't ever expect to use.

Next in importance to the Core wordset is the Core Extension wordset, section 6.2. These are *required-word wannabees and has-beens.* They make your programs easier to write and to read. They are tagged "*core ext*" in Listing One.

A.6.2 Core extension words
The words in this collection fall into several categories:
- Words that are in common use but are deemed less [sic] essential than Core words (e.g., 0<>);
- Words that are in common use but can be trivially defined from Core words (e.g., FALSE);
- Words that are primarily useful in narrowly defined types of applications or are in less frequent use (e.g. PARSE);
- Words that are being deprecated in favor of new words introduced to solve specific problems (e.g., CONVERT).

Because of the varied justifications for inclusion of these words, the Technical Committee does not encourage implementors to offer the complete collection, but to select those words deemed most valuable to their clientele.

Since my principal use of Forth is text processing, I also want the String wordset, section 17. These too are *required-word wannabees and has-beens.* They are tagged "*string*" in Listing One.

This wordset is small enough to list all of its words here.

```
-TRAILING   /STRING   BLANK   CMOVE   CMOVE>
COMPARE   SEARCH   SLITERAL
```

SLITERAL is new in Standard Forth, and may not be familiar to you.

17.6.1.2212 **SLITERAL** STRING
Interpretation: Interpretation semantics for this word are undefined.
Compilation: (c-addr1 u --)
Append the run-time semantics given below to the current definition.
Run-time: (-- c-addr2 u)
Return c-addr2 u describing a string consisting of the characters specified by c-addr1 u during compilation. A program shall not alter the returned string.

A.17.6.1.2212 **SLITERAL**
The current [Core] functionality of 6.1.2165 S" may be provided by the following definition:

```
: S" ( "ccc<quote>" -- )
   [CHAR] " PARSE   POSTPONE SLITERAL
; IMMEDIATE
```

The SLITERAL used in Stretching Forth follows the specification in the File Access wordset, 11.6.1.2165, which gives the following interpretation semantics.

Interpretation: (*"ccc<quote>" -- c-addr u*)
Parse ccc delimited by " (double quote). Store the resulting string c-addr u at a temporary location. The maximum length of the temporary buffer is implementation-dependent but shall be no less than 80 characters. Subsequent uses of S" may overwrite the temporary buffer. At least one such buffer shall be provided.

Some applications have words that have a special meaning in the context of the application. For them, Standard Forth has the Search-Order wordset, section 16.6.1. The words are tagged "*search*" in Listing One.

```
DEFINITIONS   FORTH-WORDLIST   GET-CURRENT
GET-ORDER   SEARCH-WORDLIST   SET-CURRENT
SET-ORDER   SET-CURRENT   SET-ORDER   WORDLIST
```

Listing Three. Some nominal uses of UNDER+.

```
: +!      DUP @ UNDER+ ! ;                                    ( n addr -- )
: C+!     DUP C@ UNDER+ C! ;                                ( n c_addr -- )
: COUNT   DUP C@ 1 CHARS UNDER+ ;       ( c_addr -- c_addr+1c char )
: /STRING DUP  NEGATE UNDER+  CHARS UNDER+ ;  ( s n k -- s+kc n-k )
: Count-Bits                                                 ( n -- k )
   0 SWAP BEGIN ( k n) DUP 1 AND UNDER+   2/   ?DUP 0= UNTIL   ( k)
;
```

There is also section 16.6.2, the Search-Order Extension wordset. This is the Forth-83 ONLY-ALSO experimental proposal, and can be implemented with what we already have. I don't expect to use any of them in Stretching Forth. I need only four words from these two wordsets: DEFINITIONS, FORTH-WORDLIST, SET-ORDER, WORDLIST.

I hope to have TIME&DATE from the Facilities Extension wordset.

I want some of the words from the Programming Tools wordset and Programming Tools Extension wordset — [IF], [ELSE] , [THEN] , .S, ? , and BYE — to be used *while programming*, not *in programs*. These words are given in Listing One.

For Stretching Forth, I don't anticipate needing any of the other wordsets: Double-Number, Exception, Floating-Point, Locals, Memory. If I ever use them I'll let you know.

To call attention to words originated by the Technical Committee, I have underlined them. (For text-only versions of this article, they are given in Listing Two.) Familiar words from earlier systems that were adopted by the Technical Committee are not underlined.

Every programmer has frequently used implementation factors — favorite non-Standard words that are used over and over. These are personal *required-word wannabees*. Mine have been tagged *"stock,"* for private stock. When these or any other non-Standard words (other than NOT) are used in Stretching Forth, definitions or explanations will be provided. Sample definitions are given in the Appendix. Use your own definitions if you don't like mine. In particular, you may want to replace those that use EVALUATE. Many of the sample definitions are not the ones I actually use, although functionally the same.

Logical NOT and Illogical NOT

The most controversial word in Forth is NOT. It is so controversial that the Technical Committee could not agree on standardizing it.

> A.6.1.1720 **INVERT**
> The word NOT was originally provided in Forth as a flag operator to make control structures readable. Under its intended usage the following two definitions would produce identical results:
> ```
> : ONE  ( flag -- )
>    IF ." true" ELSE ." false" THEN ;
> : TWO ( flag -- )
>    NOT IF ." false" ELSE ." true" THEN ;
> ```
> This was common usage prior to the Forth-83 [so-called] Standard which redefined NOT as a cell-wide one's-complement operation, functionally equivalent to the phrase -1 XOR. At the same time, the data type manipulated by this word was changed from a flag to a cell-wide collection of bits and the standard value for true was changed from "1" (rightmost bit only set) to "-1" (all bits set). As these definitions of TRUE and NOT were incompatible with their previous definitions, many Forth users continue to rely on the old definitions. Hence both versions are in common use.
> Therefore, usage of NOT cannot be standardized at this time. The two traditional meanings of NOT – that of negating the sense of a flag and that of doing a one's complement operation – are made available by 0= and INVERT, respectively.

Categorically, Logical NOT is the proper definition. In my understanding, the Technical Committee did not intend that NOT wasn't to be used, but that the user would provide a definition. In Stretching Forth I try to use NOT only where it doesn't matter. But for optimization, NOT should be Logical NOT.

Don't Shuffle the Stack

In writing Forth you should try to minimize re-arrangements of the stack. This is especially true in the body of a definition. Whenever you do a stack operation you have to re-visualize what is on the stack. For this reason I prefer 2DUP to OVER OVER, 1 UNDER+ to >R 1+ R> or SWAP 1+ SWAP, NIP to SWAP DROP, TUCK to SWAP OVER, BOUNDS to OVER + SWAP, NEGATE UNDER+ to SWAP >R - R>, 3DUP to DUP 2OVER ROT, 2 PICK to >R OVER R> SWAP, and so forth.

UNDER+ gives you a second cell on the stack that can be used as an accumulator without re-arranging the stack. Many algorithms can be expressed more easily with a second accumulator. It should be a primitive word, not defined in high-level Forth. The definition : UNDER+ ROT + SWAP ; will be given for UNDER+, but I hope you don't have to use it. [See Listing Three.]

Exception: I prefer ROT ROT to -ROT because it is primitive with simple optimization. Also the Technical Committee rejected -ROT when it was proposed along with NIP and TUCK. (And it makes one less definition I have to supply.)

Allergies

There are certain words in Standard Forth that I avoid using in Stretching Forth. Either they are not relevant to the general subject matter of Stretching Forth, or I use something else that I think is easier to understand.

Obsolescent words are never used.

ACCEPT and SOURCE are the latest and greatest words for source input. They replace Forth-83 EXPECT, #TIB, SPAN, and TIB — which replaced something else in Forth-79. Some day we'll get it right. I don't think we have yet. Stretching Forth will generally use WORD and PARSE

for source input.

REFILL, which can be used with WORD and PARSE, replaces QUERY. We may have gotten this one right.

>IN, SAVE-INPUT, RESTORE-INPUT, and SOURCE-ID are also used in connection with source input and system hanky-panky.

For BLANK Stretching Forth uses BL FILL; for CMOVE or CMOVE> Stretching Forth generally uses CHARS MOVE.

Instead of CASE ... OF ... ENDOF ... 0 ENDCASE Stretching Forth prefers CASE ... OF ... ELSE ... ESAC. In the future I will use all lower–case for words that replace Standard words: case ... of ... ELSE ... esac.

ENVIRONMENT? will be used for information only.

EVALUATE will generally be used instead of >NUMBER or POSTPONE. This is done for clarity of exposition. In stock definitions I may use >NUMBER or POSTPONE.

EVALUATE or POSTPONE will always be used instead of [COMPILE].

I have nothing against POSTPONE, and agree that EVALUATE could be dangerous if used wrong.

Arithmetic calculations will not be bracketed by [and] LITERAL. This kind of optimization is best left to the compiler. See the Stretching Forth article on Pinhole Optimization.

FM/MOD, SM/REM, or M* won't be used unless the definition depends on the method of division: floored or truncated.

Sensitivity

> 3.4.2 ... A system may be either case sensitive, treating upper- and lower-case letters as different and not matching, or case insensitive, ignoring differences in case while searching.

Stretching Forth code is written so that it doesn't matter whether the system is case sensitive or case insensitive. This is done by writing Standard words in upper case, and being consistent with the spelling of other words. Words are not distinguished by variations of case. I make an exception for my simplified "case" and "of," which serve the same purpose as Standard CASE and OF.

Acknowledgment

Selections from the Standard are taken from ANSI X3.215-1994, copyright © 1994 by Technical Committee X3J14.

Wil Baden is a professional programmer with an interest in Forth.
wilbaden@netcom.com

Listing One. Working words.

```
!
#  #>  #S  '  (  (.) stock
*  */  */MOD
+  +!  +LOOP
,
-  -TRAILING string
.  ."  .( core ext  .R core ext
/  /MOD  /STRING string
0<  0<>  core ext  0=  0>  core ext  0x stock
1+  1-
2!  2*  2/  2>R core ext  2@  2DROP  2DUP  2OVER  2R>  core ext
2R@ core ext  2SWAP
3DROP stock  3DUP stock
4DROP stock  4DUP stock
:  :NONAME core ext
;
<  <#  <>  core ext
=
>  >BODY  >R
?? stock  ?DO core ext  ?DUP
@
ABORT  ABORT"  ABS  AGAIN core ext  ALIGN  ALIGNED
ALLOT  AND  ANDIF stock
BASE  BEGIN  BL  BOUNDS stock
C!  C" core ext  C+! stock  C,  C@  CASE stock  CELL+  CELLS  CHAR
CHAR+  CHARS  COMPARE string  COMPILE, core ext  CONSTANT
COUNT  CR  CREATE
DECIMAL  DEPTH  DO  DOES>  DROP  DUP
ELSE  EMIT  ERASE core ext  ESAC stock  EVALUATE  EXECUTE  EXIT
FALSE core ext  FILL  FIND
HAVE stock  HERE  HEX core ext  HOLD
I  IF  IMMEDIATE  INVERT
J
KEY
LEAVE  LITERAL  LOOP  LSHIFT
MARKER core ext  MAX  MIN  MOD
MOVE  NEGATE  NIP core ext  NOT rationale
OF stock  OR  ORIF stock  OVER
PAD core ext  PARSE core ext  PARSE-WORD rationale
```

(Continues on next page.)

```
PICK core ext  PLACE stock
QUIT
R>  R@  RECURSE  REFILL core ext  REPEAT
ROLL core ext  ROT  RSHIFT
S"  S+ stock  S, stock  S= stock  S>D
SEARCH string  SIGN  SLITERAL string  SPACE  SPACES  STATE  SWAP
TH stock  THEN  TIME&DATE facility ext
TO core ext  TRUE core ext  TUCK core ext  TYPE
U.  U.R core ext  U<  U> core ext  UM*  UM/MOD  UNDER+ stock
UNLOOP  UNTIL  UNUSED core ext
VALUE core ext  VARIABLE
WHILE  WITHIN core ext  WORD
XOR
[  ['] [0x] stock  [CHAR]
\ core ext
]
.. stock  .S tools  ? tools  BYE tools ext  WORDS tools
[ELSE] tools ext  [IF] tools ext  [THEN] tools ext
DEFINITIONS search-order  FORTH-WORDLIST search-order
SET-ORDER search-order  WORDLIST search-order
INCLUDED file
```

Obsolescent

```
#TIB core ext  CONVERT core ext  EXPECT core ext  QUERY core ext
SPAN core ext  TIB core ext  FORGET tools ext
```

Avoided in Stretching Forth exposition

```
>IN
>NUMBER
ACCEPT
BLANK string  CMOVE string  CMOVE> string
CASE core ext  ENDCASE core ext  ENDOF core ext  OF core ext
ENVIRONMENT?
POSTPONE
SOURCE  SOURCE-ID core ext
SAVE-INPUT core ext  RESTORE-INPUT core ext
FM/MOD floored division
SM/REM truncated division
M*
[COMPILE] core ext

GET-CURRENT search-order  GET-ORDER search-order
SEARCH-WORDLIST search-order  SET-CURRENT search-order
```

Listing Two. Words originated by the Standard.

```
:NONAME  >NUMBER  ACCEPT  ALIGN  ALIGNED  CHAR  CHAR+  CHARS
COMPARE  COMPILE,  ENVIRONMENT?  EVALUATE  FM/MOD
FORTH-WORDLIST  GET-CURRENT  GET-ORDER  INCLUDED  INVERT  LSHIFT
MARKER  PARSE  PARSE-WORD  POSTPONE  REFILL  RESTORE-INPUT  RSHIFT
SAVE-INPUT  SEARCH  SEARCH-WORDLIST  SET-CURRENT  SET-ORDER  SLITERAL
SM/REM  SOURCE-ID  TIME&DATE  UNLOOP  UNUSED  WORDLIST  [CHAR]
[ELSE]  [IF]  [THEN]
```

Appendix. Stock definitions.

```
 ( Sample Definitions for Stock Words -- Yours may vary. )

: UNDER+  ROT + SWAP ;                          ( a b c -- a+c b )
: (.)     DUP >R   ABS   0 <#  #S   R> SIGN  #> ;  ( n -- s . )
: 3DROP   2DROP DROP ;                          ( a b c -- )
: 3DUP    2 PICK  2 PICK  2 PICK ;     ( a b c -- a b c a b c )
: 4DROP   2DROP 2DROP ;                         ( a b c d -- )
: 4DUP    2OVER 2OVER ;          ( a b c d -- a b c d a b c d )
: BOUNDS  OVER + SWAP ;                        ( a k -- a+k a )
: C+!     DUP C@ UNDER+ C! ;                      ( n a -- )
: HAVE    BL WORD FIND NIP ;                     ( -- 0|-1|1)
: PLACE   2DUP >R >R   CHAR+   SWAP CHARS MOVE ( ) R> R> C! ;
: S=      S" COMPARE 0= " EVALUATE ; IMMEDIATE
```

(Continues on next page.)

```
: S,      DUP C,   0 ?DO COUNT C, LOOP DROP ;          ( s . -- )
: ..      .S   BEGIN DEPTH WHILE DROP REPEAT ;         ( ... -- )

          ( Simplified CASE )

\ Envir. dependency on data stack used for control-flow stack.

          VARIABLE _case_sys_          ( Should be hidden. )
: case    _case_sys_ @    DEPTH _case_sys_ ! ; IMMEDIATE
: esac                              ( C: case-sys ... -- )
          BEGIN       DEPTH _case_sys_ @ >
          WHILE       postpone THEN
          REPEAT
          _case_sys_ !
; IMMEDIATE
: of
          POSTPONE OVER  POSTPONE =  POSTPONE IF  POSTPONE DROP
; IMMEDIATE
: ANDIF   POSTPONE DUP  POSTPONE IF  POSTPONE DROP ; IMMEDIATE
: ORIF    POSTPONE ?DUP  POSTPONE 0=  POSTPONE IF ; IMMEDIATE

          ( Circular String Buffer )

          2000 CONSTANT circ-Buffer-Size

          CREATE circ-Buffer  0 ,  circ-Buffer-Size CHARS ALLOT

          : Get-Buffer                                ( n -- a n)
               DUP circ-Buffer @ +  circ-Buffer-Size >
                    IF   0 circ-Buffer !   THEN
               circ-Buffer DUP @ CHARS +  CELL+          ( n a)
               SWAP DUP circ-Buffer +!                   ( a n)
          ;

: S+                                  ( s1 k1 s2 k2 -- s3 k3 )
     2 PICK OVER + Get-Buffer ( i.e. s3 k3) >R >R
          2 PICK CHARS R@ + SWAP CHARS MOVE           ( s1 k1)
          R@ SWAP CHARS MOVE                             ( )
     R> R>                                            ( s3 k3)
;

: 0x                                    ( "<spaces>name" -- ? )
     S" HEX "                                          ( s .)
     BL WORD COUNT S+
     S" DECIMAL " S+
     EVALUATE                                            ( ?)
; IMMEDIATE

: ??                                  ( n "<spaces>name" -- ? )
     S" IF "                                        ( n  s .)
     BL WORD COUNT S+
     S" THEN " S+
     EVALUATE                                            ( ?)
; IMMEDIATE

: th                                ( n "<spaces>name" -- addr )
     S" CELLS "                                     ( n  s .)
     BL WORD COUNT S+
     S" + " S+
     EVALUATE                                         ( addr)
; IMMEDIATE

: [0x]                                ( C: "<spaces>name" -- ? )
     S" [ HEX ] "                                      ( s .)
     BL WORD COUNT S+
     S" [ DECIMAL ] " S+
     EVALUATE                                            ( ?)
; IMMEDIATE

\ Procedamus in pace.
```

Review from abroad:

FORML 1995

András Zsótér
Hong Kong

I was a bit surprised when asked to write about this year's FORML conference for I am a newcomer to this event, while most of the attendees have known each other for many years. On the other hand, this might make my opinion more interesting. Of course, I can write only about my impressions, I do not know how many of the participants share my views.

Also, what can I write for *FD* when, after a couple of wines, I asked Peter Midnight what he does and he answered, "I write small things." And he described to me his tiny Forth system with a four-bit-wide stack and one-letter-long words. As a personal talk it was very nice, but what could I possibly write about it?

This was my first trip to the American continent and many things were completely new to me. I arrived at San Francisco on the afternoon of Thanksgiving (a usual complaint about the FORML conference: why does it have to be on the 24th?), which made my very first trip to the USA even more special. Next morning, John Hall and Brad Rodriguez picked me up on their way to Asilomar. On the two-hour trip, I discovered that my situation would be unusual because, as a Hungarian who happens to study in Hong Kong, I represented two continents. (We had participants from France, so I was not the only European.)

The conference started with registration and lunch, followed by the afternoon session. The most remarkable thing was that, unlike other conferences where people present boring details about work which is not understood by more than a few participants, Forthists seemed to understand each other's work very well. I was surprised how easy it was for me to follow the presentations and discussions about completely different topics.

While most of the presentations were about traditional Forth topics, including automation control, engineering and scientific applications, details about the Forth Scientific Library, and arguments about the usefulness of Forth for writing programs which can be tested in every detail (by Dr. Ting), two of them in particular drew my attention.

The first was Andrew McKewan's presentation about using Forth in the Windows NT environment and implementing an OLE automation server in Forth. I think this topic can have a huge impact on the acceptance of Forth. If we can produce applications which integrate peacefully with major operating systems, more people will accept Forth as an alternative tool.

The other unusual topic was presented by Christopher Lavaranne. He suggested a *stateless* Forth—a Forth system

(Continues on page 43.)

Using Forth Professionally?

During FORML, we had to fill in a couple of questionnaires, which, in my opinion, are not the best tools to extract information from people. Many of us claimed that if a single word were different in a question, our answer would be completely different. With one question I did not even know what to answer: "How long have you been using Forth professionally?" (Or something similar.)

I have been programming in Forth with the intention of making money (or rather, to do my research which, in turn, is my source of income) only in the last three years. On the other hand, I learned Forth about ten years ago and used its concepts in previous projects even though I did not use a Forth system.

Once I was asked to write a piece of controller software for a new instrument. At that stage, the development team already had spent one year developing the hardware, but the "software" part was only a few lines of code written in GWBASIC—barely enough to *test* the hardware. Because the rest of the software (user interface and such) was written in Turbo Pascal, I started to develop the controller program in the same language. Within two weeks, the system was up and running—something no one from that team would believe after almost a whole year of failure at developing a consistent system.

Although I actually did not use Forth as a tool in that project, my programming style was very much influenced by it. I almost wrote Forth programs in Pascal. What I mean is that I wrote tiny little procedures (almost as small as a Forth word) and tested each of them individually. Well, it would have been easier with a real Forth, but I did not have that choice.

So, what is considered to be "using Forth professionally?"

—András Zsótér

Forthware

Using Forth to manipulate the real world

Stepper Motors

Skip Carter
Monterey, California

1. Introduction

This is the first in a series of articles on using Forth to interact with the real world. We will explore how to control motors of various types (such as servomotors and stepper motors), switch power to devices, and sense the environment. Each article will present a project that can be used to demonstrate the ideas we are going to discuss.

In this first article, I want to lay the foundation for the future columns and discuss the use of the PC parallel port to control stepper motors. We will adopt the fantasy that we are working on some microprocessor-based control application and will be using the PC parallel port as a proxy for the digital I/O channels on our controller. To the extent possible, the code will be written in high level (so that we can illustrate the principles clearly), and will be in ANS Forth.

2. The PC Parallel Port

First, if you haven't already, go to your back issues of *Forth Dimensions* and find Ken Merk's article "Forth in Control" (*FD* XVII/2). In that article, Ken talks about using the PC parallel port for eight digital outputs. We will be expanding on that and use some of those other pins to get *input* as well as to provide output.

A parallel port on the PC is really three address locations which, for conventional use, could be called *#Data*, *#Command*, and *#Status*.

The port #Data is at the base address of the parallel port, #Status is at the base address plus one, and #Command is at base plus two.

The base address depends upon which parallel port we are using and the hardware installed in your computer; usually, this address is one of the hex addresses 03BC, 0378, or 0278. The BIOS determines the address and maps it to the parallel ports at boot time. This allows an application to find out where the port is by simply reading the table in memory that starts at 0040:0008. Ken shows in his article how to get this value and set a constant containing the base address for the first port; we will do the same here.

Table One. The PC parallel port.

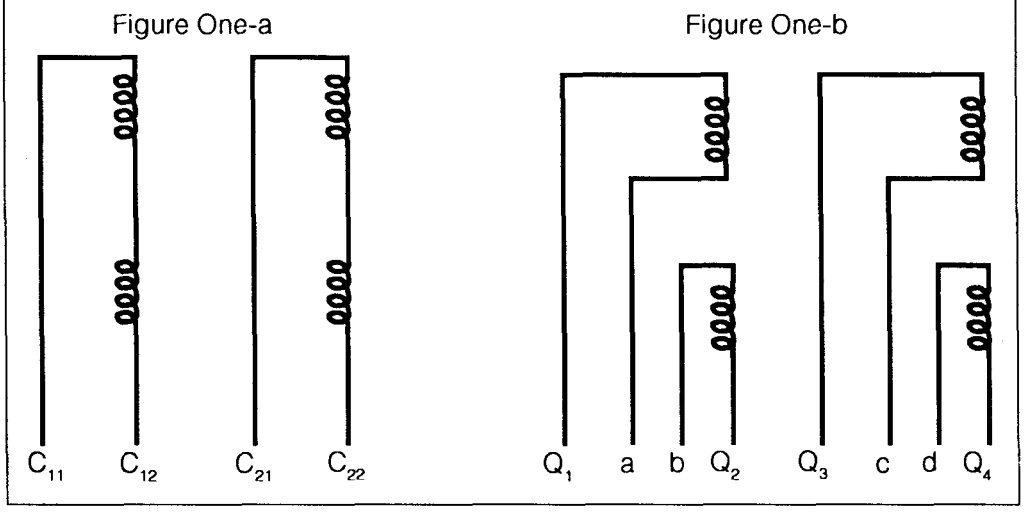
DB-25 Pin	Signal	Direction	Port	Bit
1	Strobe*	out	#Command	0
2	$Data_0$	out	#Data	0
3	$Data_1$	out	#Data	1
4	$Data_2$	out	#Data	2
5	$Data_3$	out	#Data	3
6	$Data_4$	out	#Data	4
7	$Data_5$	out	#Data	5
8	$Data_6$	out	#Data	6
9	$Data_7$	out	#Data	7
10	Ack*	in	#Status	6
11	Busy	in	#Status	7
12	Paper_out	in	#Status	5
13	Select_out	in	#Status	4
14	Auto_Feed*	out	#Command	1
15	Error*	in	#Status	3
16	Init*	out	#Command	2
17	Select_in*	out	#Command	3
18 to 25	Ground	NA	NA	NA

Table One shows what all the pins on the connector are for. You will notice that #Status port bits zero, one, and two and #Command bits five, six, and seven are not used. The #Command port is used as an output port when the port is being used for a printer, but it is actually an open-collector I/O port and can be used for input. The #Data port latches whatever was written to it, so a read from that port returns the same value that was last written to it. A single PC parallel port then gives us 12 output bits and four input bits, under normal circumstances. (Many PCs use general-purpose parallel I/O chips to implement the parallel port and can actually be programmed to be bi-directional on all the pins. Unfortunately, this form of the port is not universal.) For this project we will only need the first four data lines and ground (DB-25 pins two through five and pin 25).

3. Stepper Motors

As our first application, let us consider the control of stepper motors. Stepper motors provide *open-loop, rela-*

Figure One. (a) The internal arrangement of the coils for a bipolar stepper motor. (b) The internal arrangement of the coils for a unipolar stepper motor. Wires a through d are attached to the positive motor power supply. Six-wire motors internally connect a with b and c with d; five-wire motors internally connect a, b, c, and d.



tive motion control. Open loop means that, when you command the motor to take 42 steps, it provides no direct means of determining that it actually did so. The control is relative, meaning that there is no way to determine the shaft position directly. You can only command the motor to rotate a certain amount clockwise or counter-clockwise from its current position. These "commands" consist of energizing the various motor coils in a particular sequence of patterns. Each pattern causes the motor to move one step. Smooth motion results from presenting the patterns in the proper order.

Features that stepper motors provide include:

- Excellent rotational accuracy
- Large torque
- Small size
- Work well over a range of speeds
- Can be used for motion or position control

There are two types of stepper motors:

- *Bipolar* motors, with *two coils.* These have four wires on them (see Figure One-a). They are tricky to control because they require changing the direction of the current flow through the coils in the proper sequence. We will discuss these motors further when we get to the topic of DC motor control.
- *Unipolar* motors, with *two center-tapped coils* which can be treated as four coils (see Figure One-b). These have six or eight (or sometimes five) wires, and can be controlled from a microprocessor with little more than four transistors (see Figure Two).

Stepper motors vary in the amount of rotation delivered per step. They can turn as little as 0.72° to as much as 90° per step. The most common motors are in the 7.5° to 18° per step range. Many have integral reduction gear trains so that they have even higher angular resolution. The motor shaft will freely rotate when none of the coils are energized, but if the last pattern in a series is maintained, the motor will resist being moved to a different position. Because the motors are open-loop, if you do manage to mechanically overwhelm the motor and turn the shaft to a new position, the motor will not try to restore itself to the old position.

There are stepper motor driver ICs available, but these can be *very* expensive (as much as $20 to $50). The sequences are relatively easy to generate with a couple of TTL or CMOS chips at a much lower cost. This is the approach I typically use for most of my real stepper motor applications, since it is a good compromise between parts cost and part count, and it has a low impact on my I/O pin budget.

The easiest way to control a stepper motor is by using four bits of a parallel I/O port from a computer or microprocessor. I usually use this approach when experimenting or when the part count must be as small as possible. The microprocessor approach also has the advantage of being able to use more than one stepper sequence.

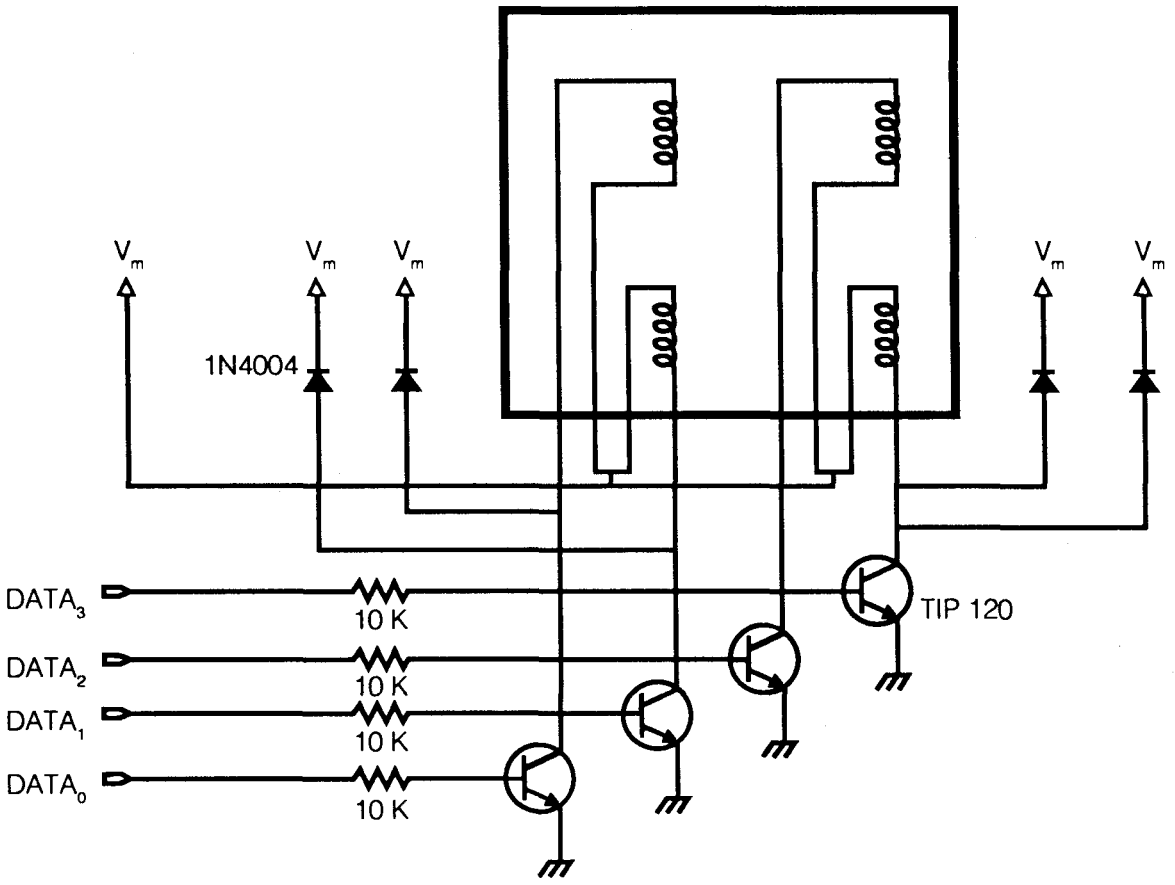
The interface to control the motor from the parallel port is just a transistor switch replicated four times. The transistor is there to control the current, which is much higher than the parallel port can sink, and to allow for the motor voltage to be independent of the PC power supply. The circuit in Figure Two can readily work with motor voltages in the range of five to 24 volts. A positive voltage at the transistor base (writing a '1' to the appropriate bit at #Data) causes the transistor to conduct. This has the effect of completing the circuit by hooking up ground to the motor coil (which has a positive voltage on the other side), so the chosen coil is turned on.

Table Two. The part list.

4	TIP 120	NPN Power Darlington transistors
4	10-K Ohm	1/4 Watt resistors
4	1N4004	Diodes
1	DB-25 Male	solder-type connector

Switching is one of the primary uses of transistors—we are using a power transistor so that we can switch lots of current (up to five amps for the TIP 120). A Darlington transistor is really a transistor pair in a single package with one transistor driving the other. A control signal on the base is amplified and then drives the second transistor. The resulting circuit can not only switch large currents, but it can do so with a very small controlling current. The

Figure Two. The interface circuit to control unipolar stepper motors from a four-bit I/O port.



resistors are to provide current limiting through the parallel port. The diodes are a feature typical of circuits that handle magnetic coils, that is *inductive* circuits. In this context, the motor windings are the inductive element. Capacitors provide a means for the storage of electrical *charge*, inductors provide a means for storage of electrical *current*. The driving current causes a magnetic field to be built up in the coil. As soon as the drive is removed, the magnetic field collapses and causes the inductor to release its stored current. Semiconductors are particularly sensitive to these currents (they briefly become conductors and then become permanent nonconductors!). The diodes provide a mechanism to safely shunt these currents away and, thus, protect the transistors and the computer. We will be seeing shunting circuits of various types in all the devices we will consider when inductive loads are involved.

The whole circuit can easily be built on a 1 7/8" by 2 3/4" prototyping board. For experimenting, it is convenient to connect the motor to the circuit via one or two feet of hookup wire with alligator clips on them instead of wiring the motor directly to the circuit. That way, a different motor can be attached to the circuit in a few seconds. You should also note that ground for the transistors must be made common between the parallel port (say at pin 25) *and* the motor power supply. An additional wire with an alligator clip can be used to provide access to the ground for the motor power supply. So, on the motor side of the circuit we have six wires, one for each coil, one for ground and one for the motor voltage on the shunt diodes. The motor (positive) voltage supply is provided through the common coils.

After building the circuit, connect the transistors Q_1 through Q_4 (via their current limiting resistors) to the DB-25 connector pins two through five. When attached to the PC parallel port, the transistors will be controlled by the low four bits of the #Data port. Don't forget the ground wire on pin 25!

3.1 Stepper motor sequencing — unipolar.

There are several kinds of sequences that can be used to drive stepper motors. The following tables give the most common sequences for energizing the coils. In all cases, the steps are repeated when reaching the end of the table. Following the steps in ascending order drives the motor in one direction, going in descending order drives the motor the other way.

Table Three. The normal sequence.

Step	Q_4	Q_3	Q_2	Q_1
1	0	1	0	1
2	1	0	0	1
3	1	0	1	0
4	0	1	1	0

Table Four. The wave drive sequence.

Step	Q_4	Q_3	Q_2	Q_1
1	0	0	0	1
2	1	0	0	0
3	0	0	1	0
4	0	1	0	0

Table Four shows what is known as the *wave drive* sequence. This sequence energizes only one coil at a time. For some motors, this sequence gives a smoother motion than the normal sequence.

Table Five. The half-step sequence.

Step	Q_4	Q_3	Q_2	Q_1
1	0	1	0	1
2	0	0	0	1
3	1	0	0	1
4	1	0	0	0
5	1	0	1	0
6	0	0	1	0
7	0	1	1	0
8	0	1	0	0

Table Five shows the *half-step* sequence. This sequence interleaves the normal and wave sequences. It *doubles* the angular resolution of the steps, so a 200-step-per-revolution motor now takes 400 steps to complete a revolution.

3.2 The bipolar sequence.

Although we will defer the discussion of bipolar stepper motors, for completeness we present the step sequence here in Table Six. These motors cannot be half-stepped.

Table Six. The bipolar sequence.

Step	C_{11}	C_{12}	C_{21}	C_{22}
1	-V	+V	-V	+V
2	-V	+V	+V	-V
3	+V	-V	+V	-V
4	+V	-V	-V	+V

3.3 Timing issues for stepper motors.

Since steppers are mechanical devices, the timing of the step pulses is important.

The motor must reach the step before the next voltage sequence is applied. If the step rate is too fast, the motor can react in one of several ways:

- it might not move at all, or
- it could vibrate in place, or
- it could rotate erratically, or
- it might rotate *in the opposite direction!*

For very smooth startups, the step rate can be started slow and gradually ramped up to a higher rate. The reverse can be done for smooth stops.

3.4 The control software.

The control code steppers.seq in Listing One can drive a motor with any of the above unipolar sequences in either direction. The code loads fcontrol.seq, from Ken Merk's article, to find the port and define the words to control the bits on the port. Several other files from the Forth Scientific Library are loaded as well: fpc2ans.seq loads an ANS-like layer on top of F-PC (a true ANS Forth would not need this), fsl-util.seq defines several utility words that are used throughout the Scientific Library, structs.seq loads the data structure words. The data structure sequence is defined to easily manage the sequence of values as defined in the sequence tables given in section 3.1 above. The sequence structures keep track of where in the sequence we are, so that there is no jump in the sequencing if one were to type `7 NORMAL STEPS`, stopped to (say) read a sensor, and then continued on with another `7 NORMAL STEPS`. This could be done with global variables instead of a data structure. However, the use of a data structure to contain this information is much more natural to extend, if the application were to require several stepper motors, than is the global variable approach.

All the software is available via anonymous FTP at taygeta.com in pub/Forth/FD.

4. The Future

In upcoming articles, we will be looking at various projects to illustrate the use of Forth to control and measure the real world. Please send your comments, suggestions, and criticisms to me through *Forth Dimensions* or via e-mail at skip@taygeta.com. In the meantime, re-tin those soldering irons!

Skip Carter is a scientific and software consultant. He is the leader of the Forth Scientific Library project, and maintains the system *taygeta* on the Internet. He is also the President of the Forth Interest Group.

Guide to listings:

STEPPPERS.SEQ (Listing One) 23–24
ANSI.SEQ 24–28
DYNMEM.SEQ 29–31
FPC2ANS.SEQ 32
FSL-UTIL.SEQ 33–37
STRUCTS.SEQ 38–41
FCONTROL.SEQ – See Merk, *Forth in Control* (*FD* XVII/2)

Listing One.

```
\ steppers.seq          code to directly drive stepper motors
\ This code is released to the public domain      September 1995
\ Taygeta Scientific Inc.

\ $Author:   skip_carter  $
\ $Workfile:   steppers.seq  $
\ $Revision:   1.0  $
\ $Date:   Oct 19 1995 11:34:12  $

fload fpc2ans.seq       \ load stuff to make it look ANS
fload fsl-util.seq
fload structs.seq       \ data structures  (V1.9 for FPC)
fload fcontrol.seq

\ ======================================================================

CR .( STEPPERS.SEQ          V1.0          22 Sept 1995  )

structure sequence
          integer:  .n
          integer:  .index
   8 integer array:  .s{
endstructure

-1 VALUE direction?

12 VALUE wtime          \ inter-step wait time (ms)

\ the data structures containing the proper coil sequences
sequence normal
sequence wave
sequence half

\ ======================================================================

: init-seqs ( -- )                \ initialize the sequences
      4 normal .n !
      0 normal .index !
      5 normal .s{ 0 } !
      9 normal .s{ 1 } !
     10 normal .s{ 2 } !
      6 normal .s{ 3 } !


      4 wave .n !
      0 wave .index !
      1 wave .s{ 0 } !
      8 wave .s{ 1 } !
      2 wave .s{ 2 } !
      4 wave .s{ 3 } !

      8 half .n !
      0 half .index !
      5 half .s{ 0 } !
      1 half .s{ 1 } !
      9 half .s{ 2 } !
      8 half .s{ 3 } !
     10 half .s{ 4 } !
      2 half .s{ 5 } !
      6 half .s{ 6 } !
      4 half .s{ 7 } !
;
```

(steppers.seq continues on next page.)

MAKE YOUR SMALL COMPUTER
THINK BIG

(We've been doing it since 1977 for IBM PC, XT, AT, PS2, and TRS-80 models 1, 3, 4 & 4P.)

FOR THE OFFICE — Simplify and speed your work with our outstanding word processing, database handlers, and general ledger software. They are easy to use, powerful, with executive-look print-outs, reasonable site license costs and comfortable, reliable support. Ralph K. Andrist, author/historian, says: "**FORTHWRITE** lets me concentrate on my manuscript, not the computer." Stewart Johnson, Boston Mailing Co., says: "We use **DATAHANDLER-PLUS** because it's the best we've seen."

MMSFORTH System Disk from $179.95
Modular pricing — Integrate with System Disk only what you need:

FORTHWRITE - Wordprocessor	$99.95
DATAHANDLER - Database	$59.95
DATAHANDLER-PLUS - Database	$99.95
FORTHCOM - for Communications	$49.95
GENERAL LEDGER - Accounting System	$250.00

FOR PROGRAMMERS — Build programs FASTER and SMALLER with our "Intelligent" MMSFORTH System and applications modules, plus the famous MMSFORTH continuing support. Most modules include source code. Ferren MacIntyre, oceanographer, says: "Forth is the language that microcomputers were invented to run."

SOFTWARE MANUFACTURERS — Efficient software tools save time and money. MMSFORTH's flexibility, compactness and speed have resulted in better products in less time for a wide range of software developers including Ashton-Tate, Excalibur Technologies, Lindbergh Systems, Lockheed Missile and Space Division, and NASA-Goddard.

MMSFORTH V2.4 System Disk from $179.95
Needs only 24K RAM compared to 100K for BASIC, C, Pascal and others. Convert your computer into a Forth virtual machine with sophisticated Forth editor and related tools. This can result in 4 to 10 times greater productivity.

Modular pricing — Integrate with System Disk only what you need:

EXPERT-2 - Expert System Development	$89.95
FORTHCOM - Flexible data transfer	$49.95
UTILITIES - Graphics, 8087 support and other facilities.	

and a little more!

THIRTY-DAY FREE OFFER — **Free MMSFORTH GAMES DISK** worth $39.95, with purchase of MMSFORTH System. CRYPTOQUOTE HELPER, OTHELLO, BREAKFORTH and others.

Call for free brochure, technical info or pricing details.

MMSFORTH

MILLER MICROCOMPUTER SERVICES
61 Lake Shore Road, Natick, MA 01760
(508/653-6136, 9 am - 9 pm)

```
: idx++  ( seq_hdl -- idx )      \ increment the index, return old value
         2DUP .n @ >R
              .index DUP @
         DUP 1+ R> MOD
         ROT !
;
: idx-- ( seq_hdl -- idx )       \ decrement the index, return old value
        2DUP .n @ >R
             .index DUP @
        DUP 1- R> OVER
        0 < IF 1- SWAP THEN DROP
        ROT !
;
: fsteps ( seq_hdl n -- )
     0 DO wtime MS
          2DUP idx++ >R
          2DUP .s{ R> } @ WRITE
       LOOP

    2DROP
;
: rsteps ( seq_hdl n -- )
     0 DO wtime MS
          2DUP idx-- >R
          2DUP .s{ R> } @ WRITE
       LOOP

    2DROP
;
\ ================================================================

: reverse  ( -- )                        \ toggles rotation direction
   direction?  IF 0 ELSE -1 THEN
   TO direction?
;

: steps ( n seq_hdl -- )                 \ take n steps from given sequence
    ROT
    direction? IF fsteps ELSE rsteps THEN
;
\ ================================================================
init-seqs

\   To run use a sequence like:    12 NORMAL STEPS
\ end of steppers.seq
```

ANSI.SEQ

```
\ ANSI.SEQ  Provide ANSI CORE and CORE EXT by U.Hoffmann

Only Forth also Hidden also definitions   warning off

: COMP:  \ Warn if word is used in compile state
  Create , immediate
  Does> @
    state @
    IF  ." warning: outer interpreter not ANS compliant!" cr
        X,  EXIT THEN
    EXECUTE ;

: token ( -- xt )
   \ Construct all necessary data for a new ananymous colon def
   \ but do not start to compile the body itself.
   \ Return the newly created execution token.
```

(ansi.seq continues on next page.)

```
   \ Token is a factor of :NONAME and :
   here
   !CSP   233 c, ( jmp )  >nest here 2+ - ,
   XHERE PARAGRAPH +  DUP XDPSEG !  XSEG @ - ,  XDP OFF ;

: colon ( -- ) \ : without current @ context !
   HEADER  token  drop  !CSP HIDE (]) ;

Only Forth also Hidden also Forth also definitions
Vocabulary ANSI  ANSI definitions
Vocabulary CORE  ANSI CORE definitions

' !             Alias !                  ( x a-addr -- )
' #             Alias #                  ( ud1 -- ud2 )
' #>            Alias #>                 ( xd -- c-addr u)
' #S            Alias #S                 ( ud1 -- ud2 )
' '             Alias '                  ( "name" -- xt )
' (             Alias ( immediate        ( "ccc<paren>" -- )
' *             Alias *                  ( n1|u1 n2|u2 -- n3|u3 )
' */            Alias */                 ( n1 n2 n3 -- n4 )
' */MOD         Alias */MOD              ( n1 n2 n3 -- n4 n5 )
' +             Alias +                  ( n1|u1 n2|u2 -- n3|u3 )
' +!            Alias +!                 ( n|u a-addr -- )
' +LOOP         Alias +LOOP immediate    ( C: do-sys -- )
                                        ( n -- ) ( R: loop-sys11 -- | loop-sys2 )
' ,             Alias ,                  ( x -- )
' -             Alias -                  ( n1|u1 n2|u2 -- n3|u3 )
' .             Alias .                  ( n -- )
' ."            Alias ." immediate       ( "ccc<quote>" -- )
' /             Alias /                  ( n1 n2 -- n3 )
' /MOD          Alias /MOD               ( n1 n2 -- n3 n4 )
' 0<            Alias 0<                 ( n -- flag )
' 0=            Alias 0=                 ( x -- flag )
' 1+            Alias 1+                 ( n1|u1 -- n2|u2 )
' 1-            Alias 1-                 ( n1|u1 -- n2|u2 )
' 2!            Alias 2!                 ( x1 x2 a-addr -- )
' 2*            Alias 2*                 ( x1 -- x2 )
' 2/            Alias 2/                 ( x1 -- x2 )
' 2@            Alias 2@                 ( a-addr -- x1 x2 )
' 2DROP         Alias 2DROP              ( x1 x2 -- )
' 2DUP          Alias 2DUP               ( x1 x2 -- x1 x2 x1 x2 )
' 2OVER         Alias 2OVER              ( x1 x2 x3 x4 -- x1 x2 x3 x4 x1 x2 )
' 2SWAP         Alias 2SWAP              ( x1 x2 x3 x4 -- x3 x4 x1 x2 )
' colon         Comp: :                  ( "name" -- colon-sys )
                                        ( Initiation: i*x -- i*x ) ( R: -- nest-sys )
                                        ( "name" execution: i*x -- j*x )
' ;             Alias ; immediate        ( C: colon-sys -- )
                                        ( -- ) ( R: nest-sys -- )
' <             Alias <                  ( n1 n2 -- flag )
' <#            Alias <#                 ( -- )
' =             Alias =                  ( x1 x2 -- flag )
' >             Alias >                  ( n1 n2 -- flag )
' >BODY         Alias >BODY              ( xt -- a-addr )
' >IN           Alias >IN                ( -- a-addr )

: >number ( ud1 c-addr1 u1 -- ud2 c-addr2 u2 )
    BEGIN dup
    WHILE ( ud c-addr u )
      >r dup >r c@  ( ud char )
      base @ digit 0= IF  drop  r> r> ( ud c-addr u ) EXIT THEN  ( ud digit )
      swap  ( hi ) base @ um*  drop    rot ( lo ) base @ um* d+
      double?  IF   dpl incr  THEN
      r> r>  1 /string
    REPEAT ;

' >R            Alias >R                 ( x -- ) ( R: -- x )
' ?DUP          Alias ?DUP               ( x -- 0|x x )
```

(ansi.seq continues on next page.)

```
' @             Alias @                  ( a-addr -- x )
' ABS           Alias ABS                ( n -- +n )
' ABORT         Alias ABORT              ( i*x -- ) ( R: j*x -- )
' ABORT"        Alias ABORT" immediate   ( C: "ccc<quote>" -- )
                                         ( i*x flag -- i*x | ) ( R: j*x -- j*x | )
: ACCEPT ( c-addr +n1 -- +n2 )  EXPECT  SPAN @ ;
' NOOP          Alias ALIGN immediate    ( -- )
' NOOP          Alias ALIGNED immediate  ( -- )
' ALLOT         Alias ALLOT              ( n -- )
' AND           Alias AND                ( x1 x2 -- x3 )
' BASE          Alias BASE               ( -- a-addr )
' BEGIN         Alias BEGIN immediate    ( C: -- dest ) ( -- )
' BL            Alias BL                 ( -- char )
' C!            Alias C!                 ( char c-addr -- )
' C,            Alias C,                 ( char -- )
' C@            Alias C@                 ( c-addr -- char )
' 2+            Alias CELL+              ( a-addr1 -- a-addr2 )
' 2*            Alias CELLS              ( n1 -- n2 )
: CHAR ( "name" -- char )  BL WORD 1+ C@ ;
' 1+            Alias CHAR+              ( c-addr1 -- c-addr2 )
' NOOP          Alias CHARS immediate    ( n1 -- n2 )
' CONSTANT      Alias CONSTANT           ( x "name" -- ) ( -- x )
' COUNT         Alias COUNT              ( c-addr1 -- c-addr2 u )
' CR            Alias CR                 ( -- )
' CREATE        Alias CREATE             ( "name" -- ) ( -- a-addr )
' DECIMAL       Alias DECIMAL            ( -- )
' DEPTH         Alias DEPTH              ( -- +n )
' DO            Alias DO immediate       ( C: -- do-sys ) ( n1|u1 n2|u2 -- ) ( R: loop-sys )
' DOES>         Alias DOES> immediate    ( C: colon-sys1 -- colon-sy2 )
                                         ( -- ) ( R: nest-sys1 -- )
                                         ( Initiation: i*x -- i*x a-addr ) ( R: -- nest-sys )
                                         ( "name" execution: i*x -- j*x )
' DROP          Alias DROP               ( x -- )
' DUP           Alias DUP                ( x -- x x )
' ELSE          Alias ELSE immediate     ( C: orig1 -- orig2 )
' EMIT          Alias EMIT               ( x -- )
: ENVIRONMENT? ( x-addr u -- false | i*x true )   2DROP FALSE ;

Only Forth also definitions  needs eval
Hidden also ANSI CORE definitions

' EVAL Alias EVALUATE      ( i*x c-addr u -- j*x )
' EXECUTE       Alias EXECUTE            ( i*x xt -- j*x )
' EXIT          Alias EXIT               ( -- ) ( R: nest-sys -- )
' FILL          Alias FILL               ( c-addr u char -- )
' FIND          Alias FIND               ( c-addr -- c-addr 0 | xt 1 | xt -1 )
' M/MOD         Alias FM/MOD             ( d1 n1 -- n2 n3 )
' HERE          Alias HERE               ( -- addr )
' HOLD          Alias HOLD               ( char -- )
' I             Alias I                  ( -- n|u ) ( R: loop-sys -- loop-sys )
' IF            Alias IF immediate       ( C: -- orig ) ( x -- )
' IMMEDIATE     Alias IMMEDIATE          ( -- )
' NOT           Alias INVERT             ( x1 -- x2 )
' J             Alias J                  ( -- n|u ) ( R: loop-sys -- loop-sys )
' KEY           Alias KEY                ( -- char )
' LEAVE         Alias LEAVE immediate    ( -- ) ( R: loop-sys -- )
' LITERAL       Alias LITERAL immediate  ( C: x -- ) ( -- x )
' LOOP          Alias LOOP immediate     ( C: do-sys -- ) ( -- )
                                         ( R: loop-sys1 -- | loop-sys2 )
\ : LSHIFT ( x1 u -- x2 )  16 umin 0 ?DO 2* LOOP ;
CODE LSHIFT ( x1 u -- x2 ) POP CX   POP AX   SHL AX, CL   1PUSH end-code  \  courtesy akg
' *D            Alias M*                 ( n1 n2 -- d )
' MAX           Alias MAX                ( n1 n2 -- n3 )
' MIN           Alias MIN                ( n1 n2 -- n3 )
' MOD           Alias MOD                ( n1 n2 -- n3 )
' MOVE          Alias MOVE               ( addr1 addr2 u -- )
' NEGATE        Alias NEGATE             ( n1 -- n2 )
' OR            Alias OR                 ( x1 x2 -- x3 )
```

(ansi.seq continues on next page.)

```
' OVER          Alias OVER              ( x1 x2 -- x1 x2 x1 )

: POSTPONE ( C: "name" -- ) ( -- )
   state @ 0= Abort" compile only"
   defined dup 0= ?missing 0< IF ( non imm )  compile compile THEN
   X, ; immediate

' QUIT          Alias QUIT              ( -- ) ( R: i*x -- )
' R>            Alias R>                ( -- x ) ( R: x -- )
' R@            Alias R@                ( -- x ) ( R: x -- x )
' RECURSE       Alias RECURSE immediate ( C: -- )
' REPEAT        Alias REPEAT immediate  ( C: orig dest -- ) ( -- )
' ROT           Alias ROT               ( x1 x2 x3 -- x2 x3 x1 )
\ : RSHIFT ( x1 u -- x2 )  16 umin 0 ?DO u2/ LOOP ;
CODE RSHIFT ( x1 u -- x2 )  POP CX   POP AX   SHR AX, CL  1PUSH end-code \ courtesy akg
' "             Alias S" immediate      ( C: "ccc<quote>" -- ) ( -- c-addr u )
' S>D           Alias S>D               ( n -- d )
' SIGN          Alias SIGN              ( n -- )

: SM/REM ( d1 n2 -- n2 n3 )
   over >r  2dup xor 0< >r
   abs >r  dabs r>   um/mod ( rem quot )
   r> ?negate swap  r> ?negate swap ;

' SOURCE        Alias SOURCE            ( -- c-addr u )
' SPACE         Alias SPACE             ( -- )
' SPACES        Alias SPACES            ( n -- )
' STATE         Alias STATE             ( -- a-addr )
' SWAP          Alias SWAP              ( x1 x2 -- x2 x1 )
' THEN          Alias THEN immediate    ( C: orig -- ) ( -- )
' TYPE          Alias TYPE              ( c-addr u -- )
' U.            Alias U.                ( u -- )
' U<            Alias U<                ( u1 u2 -- flag )
' UM*           Alias UM*               ( u1 u2 -- ud )
' UM/MOD        Alias UM/MOD            ( ud u1 -- u2 u3 )
Code UNLOOP ( -- ) ( R: loop-sys -- )  ADD RP, # 6  NEXT   end-code
' UNTIL         Alias UNTIL immediate   ( C: dest -- ) ( x -- )
' VARIABLE      Alias VARIABLE          ( "name" -- ) ( "name" Execution: -- a-addr )
' WHILE         Alias WHILE immediate   ( C: dest -- orig dest ) ( x -- )
' WORD          Alias WORD              ( char "<chars>ccc<char>" -- c-addr )
' XOR           Alias XOR               ( x1 x2 -- x3 )
' [             Alias [ immediate       ( -- )
' [']           Alias ['] immediate     ( -- xt )
: [CHAR] ( C: "name" -- ) ( -- char )  CHAR [compile] Literal ; immediate
' ]             Comp: ]                 ( -- )

Vocabulary EXT  EXT definitions

' #TIB          Alias #TIB              ( -- a-addr )
' .(            Alias .( immediate      ( "ccc<paren>" -- )
' .R            Alias .R                ( n1 n2 -- )
' 0<>           Alias 0<>               ( x -- flag )
' 0>            Alias 0>                ( n -- flag )
' 2>R           Alias 2>R               ( x1 x2 -- ) ( R: -- x1 x2 )
' 2R>           Alias 2R>               ( -- x1 x2 ) ( R: x1 x2 -- )
' 2R@           Alias 2R@               ( -- x1 x2 ) ( R: x1 x2 -- x1 x2 )

: :NONAME ( -- colon-sys ) ( -- xt )
   token  (]) ;

' <>            Alias <>                ( x1 x2 -- flag )
' ?DO           Alias ?DO immediate     ( C: -- do-sys ) ( n1|u1 n2|u2 -- )
                                        ( R: -- | loop-sys )
```

(ansi.seq continues on next page.)

```
' AGAIN          Alias AGAIN immediate    ( C: dest -- ) ( -- )
: C" ( "ccc<quote>" -- ) ( -- c-addr )  [compile] "  compile ">$ ; immediate
' CASE           Alias CASE immediate     ( C: -- case-sys ) ( -- )
' X,             Alias COMPILE,           ( xt -- )
' CONVERT        Alias CONVERT            ( ud1 c-addr1 -- ud2 c-addr2 )
: ENDCASE ( C: case-sys -- ) ( x -- )
   compile drop  [compile] ENDCASE ; immediate
' ENDOF          Alias ENDOF immediate    ( C: case-sys1 of-sys -- case-sys2 ) ( -- )
' ERASE          Alias ERASE              ( addr u -- )
' EXPECT         Alias EXPECT             ( c-addr +n -- )
' FALSE          Alias FALSE              ( -- false )
' HEX            Alias HEX                ( -- )

: MARKER ( "name" -- )
   Create
   context  #vocs 0 DO  dup @ , 2+ LOOP drop  current @ ,
  Does> ( -- )
   dup  context  #vocs 0 DO  over @ over !  2+ swap 2+ swap LOOP drop
      @ current !
   body> dup >view (frget) ;

' NIP            Alias NIP                ( x1 x2 -- x2 )
' OF             Alias OF immediate       ( C: -- of-sys ) ( x1 x2 -- | x1 )
' PAD            Alias PAD                ( -- c-addr )
' PARSE          Alias PARSE              ( char "ccc<char>" -- c-addr u )
' PICK           Alias PICK               ( xu ... x1 x0 u -- u ... x1 x0 xu )
' QUERY          Alias QUERY              ( -- )
( REFILL is not defined )

: RESTORE-INPUT ( x1 ... xn n -- flag )
   dup 7 <> IF  0 ?DO drop LOOP  true EXIT THEN
   drop
   !> run  !> loadline  !> #tib  !> >in  !> 'tib  !> loading  !> iblen
   false ;

' ROLL           Alias ROLL               ( xu xu-1 ... x0 u -- xu-1 ... x0 xu )

: SAVE-INPUT ( -- x1 ... xn n )
   @> iblen  @> loading  @> 'tib  @> >in  @> #tib  @> loadline  @> run  7 ;

( SOURCE-ID is not defined )
' SPAN           Alias SPAN               ( -- a-addr )
' TIB            Alias TIB                ( -- c-addr )
' =:             Alias TO immediate       ( x "name" -- )  ( C: "name" -- ) ( x -- )
' TRUE           Alias TRUE               ( -- true )
' TUCK           Alias TUCK               ( x1 x2 -- x2 x1 x2 )
' U.R            Alias U.R                ( u n -- )
' U>             Alias U>                 ( u1 u2 -- flag )
: UNUSED ( -- u )   sp@ here - ;
' VALUE          Alias VALUE
: WITHIN ( n1|u2 n2|u2 n3|u3 -- flag )   over - >r - r> u< ;
' [COMPILE]      Alias [COMPILE] immediate ( "name" -- )
' \              Alias \ immediate        ( "ccc<eol>" -- )

Only Forth also Root definitions

' ANSI Alias ANSI

Only Forth also definitions  warning on

cr .( ANS-Forth compatibilty package loaded )

\S
```

DYNMEM.SEQ

```
\ dynmem.seq                  Dynamic Memory Allocation package
\                        this code is an adaptation of the routines by
\         Dreas Nielson, 1990; Dynamic Memory Allocation;
\         Forth Dimensions, V. XII, No. 3, pp. 17-27
\
\ This is an ANS Forth program requiring:
\      1. The Memory-Allocation wordset, or the implementations below of ALLOCATE and FREE
\      2. The compilation of the local ALLOCATE and FREE is controlled by
\         the VALUE HAS-MEMORY-WORDS?
\         and the conditional compilation words in the Programming-Tools wordset
\
\ This code is designed to work in conjunction with the FSL implementation
\ of arrays as given in the file, 'fsl-util'.
\
\ The words ALLOCATE and FREE are implementations of the ANS Forth
\ words from the Memory-Allocation wordset.  If your Forth system
\ has the Memory-Allocation wordset the following words can be eliminated from here:
\       freelist
\       Dynamic-Mem
\       ALLOCATE
\       FREE
\
\  To use dynamic memory, a dynamic memory pool needs to be created and
\  initialized. The dynamic memory pool needs to be initialized before it is ever
\  used.  IF THIS IS NOT DONE, ALLOCATE will abort with a message
\  complaining about the lack of initialization.  Typically
\  the initialization would look like,
\ CREATE POOL   #bytes  ALLOT
\ POOL #bytes Dynamic-Mem
\
\ (any other way of allocating space for the pool will also work, one just has to pass
\ the starting address of some contigous memory and the number of bytes to Dynamic-Mem).
\ If there are alignment requirements for the data space, this should be satisfied BEFORE
\ the address is passed to Dynamic-Mem.
\
\ If your application ends up using more bytes than are in the memory
\ pool ( #bytes ) then the internal pointer will be NULL when }malloc
\ fails.  You can detect this by invoking  malloc-fail?,
\ malloc-fail?
\
\ If there is a true on the stack at this point, then the allocation
\ failed. This allows the following usage,
\ malloc-fail? ABORT" ALLOCATE failed "
\
\ The allocation and freeing of dynamic memory can be done in any order.
\ Since this can be done in any order, there is a possiblity that the
\ pool will become fragemented.   It is then possible that a }malloc
\ will fail if the memory pool is very fragmented.
\
\ The current version of the dynamic memory package can have only one memory pool.
\
\ For dynamically allocated arrays, the delcaration looks like,
\ element_size DARRAY name{
\
\ where element_size is the number of cells that the data type occupies
\ just as for static arrays.
\
\ To allocate space for a dynamic array (this can be done at runtime),
\ & name{ #elements }malloc
\
\ If it succeeds then there will have been contiguous space allocated
\ for the required number of elements.
\
\ To release the space (this can also be done at runtime) use,
\ & name{ }free
\
\ A dynamic array name can be re-used by calling }free to release
\ the old space and then calling }malloc again to reallocate it.
```

(dynmem.seq continues on next page.)

```
CR .( DYNMEM            V1.9            4 January  1995   EFC )

Private:

HAS-MEMORY-WORDS? 0= [IF]

\ pointer to beginning of free space
variable freelist  0 ,     0 freelist !

[THEN]

Public:

\ memory allocation status variable, 0 for OK
0 VALUE malloc-fail?

: cell_size ( addr -- n )      >BODY CELL+ @ ;       \ gets array cell size

HAS-MEMORY-WORDS? 0= [IF]

\ initialize memory pool at ALIGNED address 'start_addr'
: Dynamic-Mem ( start_addr length -- )
          OVER DUP freelist !
          0 SWAP !
          SWAP CELL+ !
;

: ALLOCATE ( u -- addr ior )      \ allocate n bytes, return pointer to block
                                  \ and result flag ( 0 for success )
         \ check to see if pool has been initialized first
         freelist @ 0= ABORT" ALLOCATE::memory pool not initialized! "

         CELL+ freelist DUP
         BEGIN
           WHILE DUP @ CELL+ @ 2 PICK U<
                 IF @ @ DUP    \ get new link
                 ELSE   DUP @ CELL+ @ 2 PICK - 2 CELLS MAX DUP 2 CELLS =
                        IF DROP DUP @ DUP @ ROT !
                        ELSE  OVER OVER SWAP @ CELL+ !   SWAP @ +
                        THEN
                        OVER OVER ! CELL+ 0       \ store size, bump pointer
                 THEN                             \ and set exit flag
          REPEAT

          SWAP DROP

          DUP 0=
;

: FREE ( ptr -- ior )           \ free space at ptr, return status ( 0 for success )
           1 CELLS - DUP @ SWAP OVER OVER CELL+ ! freelist DUP
           BEGIN
             DUP 3 PICK U< AND
           WHILE
             @ DUP @
           REPEAT

           DUP @ DUP 3 PICK ! ?DUP
           IF DUP 3 PICK 5 PICK + =
              IF DUP CELL+ @ 4 PICK + 3 PICK CELL+ ! @ 2 PICK !
              ELSE   DROP THEN
           THEN

           DUP CELL+ @ OVER + 2 PICK =
           IF  OVER CELL+ @ OVER CELL+ DUP @ ROT + SWAP ! SWAP @ SWAP !
           ELSE !
           THEN
```

(dynmem.seq continues on next page.)

```
          DROP
          0           \ this code ALWAYS returns a success flag
;

[THEN]

\ word for allocation of a dynamic 1-D array memory
\ typical usage:  & a{ #elements }malloc
                                    \ ---------------------
: }malloc ( addr n -- )             \ | size | data area
                                    \ ---------------------
          OVER cell_size DUP >R *       \ save extra cell_size on rstack
          \ now add space for the cell_size entry
          CELL+ ALLOCATE
          TO malloc-fail?
          OVER >BODY !

          \ now store the cell size in the beginning of the block
          >BODY @ R> SWAP !
;

\ word to release dynamic array memory, typical usage:  & a{ }free

: }free   ( addr -- )
        >BODY DUP
        @ FREE
        TO malloc-fail?
        0 SWAP !
;

\ word for allocation of a dynamic 2-D array memory
\ typical usage:  & a{{ #rows #cols }}malloc
                                    \  -------------------------
: }}malloc ( addr n m -- )          \  | m | size | data area
                                    \  -------------------------
          2 PICK cell_size DUP
          >R OVER >R          \ save extra cell_size and m on rstack
          * *                 \ calculate the space needed
          \ now add space for the cell_size entry and m
          CELL+ CELL+ ALLOCATE
          TO malloc-fail?

          SWAP OVER CELL+ SWAP >BODY !    \ store pointer to allocated space
                                          \ Note: pointing to size field not
          \ now store m and cell size in the beginning of the block
          R> OVER !
          R> SWAP CELL+ !
;

: }}free    }free ;

Reset_Search_Order
```

FPC2ANS.SEQ

```
\ fpc2ans.seq           Loads the stuff to add ANS compliance for F-PC

cr .( FPC2ANS.SEQ       V1.6           23 September 1994     EFC )

fload ansi.seq

Only Forth also definitions ANSI CORE also

: GET-CURRENT ( -- wid )
        CURRENT @
;

: SET-CURRENT ( wid -- )
        CURRENT !
;

fload ffloat.seq        \ get the floating point package

: to      ' >body state @ if [compile] literal compile !
                          else ! then   ; immediate

: d>s     drop ;

: d>f     float ;

: F>D     int ;

: >float drop 1- (mantissa) (exp) float falog f* -1 ;

\ size of a floating point element
8 constant fcell

\ : floats   fcell *   ;

\ : cell+    1 cells + ;
: float+   1 floats + ;

: FE.   F.  ;           \ not exactly right but good enough
: SF@   F@  ;
: SF!   F!  ;

: FALIGN    ALIGN  ;
: FALIGNED  ALIGNED  ;

: FSINCOS   FDUP FSIN FSWAP FCOS ;

: FATAN2    F/ FATAN ;


' #IF Alias   [IF]
' #ELSE Alias [ELSE]
' #THEN Alias [THEN]
```

FSL-UTIL.SEQ

```
\ fsl_util.seq          An auxilliary file for the Forth Scientific Library
\                       contains commonly needed definitions.
\                       For F-PC V3.6

\ $Workfile:   fsl_util.seq  $
\ $Revision:   1.18  $
\ $Date:   27 Jan 1995 22:29:26  $

\ dxor, dor, dand       double xor, or, and
\ sd*                   single * double = double_product
\ v: defines use( &     For defining and settting execution vectors
\ %                     Parse next token as a FLOAT
\ S>F  F>S              Conversion between (single) integer and float
\ F,                    Store FLOAT at (aligned) HERE
\ INTEGER, DOUBLE, FLOAT   For setting up ARRAY types
\ ARRAY DARRAY              For declaring static and dynamic arrays
\ }                         For getting an ARRAY or DARRAY element address
\ &!                        For storing ARRAY aliases in a DARRAY
\ PRINT-WIDTH               The number of elements per line for printing arrays
\ }FPRINT                   Print out a given array
\ Matrix                    For declaring a 2-D array
\ }}                        gets a Matrix element address
\ Public: Private: Reset_Search_Order   controls the visibility of words
\ |frame frame|             sets up/removes a local variable frame
\ a b c d e f g h           local FVARIABLE values
\ &a &b &c &d &e &f &g &h   local FVARIABLE addresses

\ This code conforms with ANS requiring:
\      1. The Floating-Point word set
\      2. The words umd* umd/mod and d* are implemented
\         for F-PC in the file dmuldiv.seq
\      3. The word VOCABULARY is defined

\ This code is released to the public domain Everett Carter July 1994

CR .( FSL_UTIL.SEQ      V1.18          26 January 1995   EFC )

\ ====================== compilation control ============================
\ for control of conditional compilation test code
FALSE VALUE TEST-CODE?
FALSE VALUE ?TEST-CODE           \ obsolete, for backward compatiblity

\ for control of conditional compilation of Dynamic Memory
FALSE CONSTANT HAS-MEMORY-WORDS?

\ ======================================================================

\ FSL Non ANS words

: ~DEFINED   ( c-addr -- t/f )     \ returns true if NOT defined
     DEFINED 0=
;

\ umd/mod    ( uquad uddiv -- udquot udmod ) unsigned quad divided by double
\ umd*       ( ud1 ud2 -- qprod )            unsigned double multiply
\ d*         ( d1 d2   -- dprod )            double multiply

\ For F-PC the above three are already defined in DMULDIV.SEQ
\ needs dmuldiv.seq       \ needs definitions of umd* umd/mod and d*

: dxor       ( d1 d2 -- d )             \ double xor
     ROT XOR -ROT XOR SWAP
;

: dor        ( d1 d2 -- d )             \ double or
     ROT OR -ROT OR SWAP
;
```

(fsl-util.seq continues on next page.)

```
: dand      ( d1 d2 -- d )                \ double and
     ROT AND -ROT AND SWAP
;

\ single * double = double
: sd*    ( multiplicand  multiplier_double  -- product_double  )
             2 PICK * >R    UM*    R> +
;

\ : D0<         NIP 0< ;

: T*          TUCK UM* 2SWAP UM* SWAP >R 0 D+ R> ROT ROT ;
: T/          DUP >R UM/MOD ROT ROT R> UM/MOD NIP SWAP ;

: m*/         >R T* R> T/ ;

\ function vector definition
: v: CREATE ['] noop , DOES> @ EXECUTE ;
: defines    ' >BODY STATE @ IF [COMPILE] LITERAL COMPILE !
                              ELSE ! THEN ;    IMMEDIATE

: use(  STATE @ IF [COMPILE] ['] ELSE ' THEN ;  IMMEDIATE
: &     [COMPILE] use( ; IMMEDIATE

\ pushes following value to the float stack
: %    BL WORD  COUNT  >FLOAT  0= ABORT" NAN"
        STATE @  IF POSTPONE FLITERAL  THEN  ; IMMEDIATE

: S>F    ( n -- ) ( f: -- x )     \ integer to float
        S>D  D>F
;

: F>S     ( -- n ) ( f: x -- )    \ float to integer
        F>D DROP
;

\ Store float at (aligned) HERE
\ already defined in F-PC
\ : F,    ( -- | f: x -- )          FALIGN HERE  1 FLOATS ALLOT F!  ;

\ : F=     F- F0= ;
\ : -FROT     FROT FROT ;
\ : F2*    % 2.0e0 F*      ;
\ : F2/    % 2.0e0 F/      ;
\ : F2DUP      FOVER FOVER ;
\ : F2DROP     FDROP FDROP ;

: CELL-     [ 1 CELLS ] LITERAL - ;           \ backup one cell

0 VALUE TYPE-ID                \ for building structures
FALSE VALUE STRUCT-ARRAY?

\ for dynamically allocating a structure or array

TRUE  VALUE is-static?      \ TRUE for statically allocated structs and arrays
: dynamic ( -- )      FALSE TO is-static? ;

\ size of a regular integer
1 cells CONSTANT INTEGER

\ size of a double integer
2 cells CONSTANT DOUBLE

\ size of a regular float
1 floats CONSTANT FLOAT
```

(fsl-util.seq continues on next page.)

```
\ 1-D array definition
\    ----------------------------
\    | cell_size | data area     |
\    ----------------------------

: MARRAY ( n cell_size -- | -- addr )            \ monotype array
    CREATE
      DUP , * ALLOT
    DOES> CELL+
;

\    ----------------------------
\    | id | cell_size | data area |
\    ----------------------------

: SARRAY ( n cell_size -- | -- id addr )          \ structure array
    CREATE
      TYPE-ID ,
      DUP , * ALLOT
    DOES> DUP @ SWAP [ 2 CELLS ] LITERAL +
;

: ARRAY
    STRUCT-ARRAY? IF   SARRAY FALSE TO STRUCT-ARRAY?
                  ELSE MARRAY
                  THEN
;

\ word for creation of a dynamic array (no memory allocated)

\ Monotype
\    ------------------------
\    | data_ptr | cell_size |
\    ------------------------

: DMARRAY   ( cell_size -- )   CREATE  0 , ,
                               DOES>
                                     @ CELL+
;

\ Structures
\    ----------------------------
\    | data_ptr | cell_size | id |
\    ----------------------------

: DSARRAY   ( cell_size -- )  CREATE  0 , , TYPE-ID ,
                              DOES>
                                    DUP [ 2 CELLS ] LITERAL + @ SWAP
                                    @ CELL+
;

: DARRAY   ( cell_size -- )
    STRUCT-ARRAY? IF   DSARRAY FALSE TO STRUCT-ARRAY?
                  ELSE DMARRAY
                  THEN
;

\ word for aliasing arrays,
\  typical usage:  a{ & b{ &!  sets b{ to point to a{'s data

: &!    ( addr_a &b -- )
        SWAP CELL- SWAP >BODY  !
;

: }   ( addr n -- addr[n])       \ word that fetches 1-D array addresses
         OVER CELL- @
         * SWAP +
;
```

(fsl-util.seq continues on next page.)

```
VARIABLE print-width     6 print-width !

: }fprint ( n addr -- )        \ print n elements of a float array
        SWAP 0 DO I print-width @ MOD 0= I AND IF CR THEN
                  DUP I } F@ F. LOOP
        DROP
;

: }iprint ( n addr -- )        \ print n elements of an integer array
       SWAP 0 DO I print-width @ MOD 0= I AND IF CR THEN
                 DUP I } @ . LOOP
       DROP
;

: }fcopy ( n &src &dest -- )     \ copy one array into another
         ROT 0 DO
                 OVER I } F@
                 DUP  I } F!
               LOOP
         2DROP
;

\ 2-D array definition,
\ Monotype
\    -----------------------------------
\    | m | cell_size |  data area |
\    -----------------------------------

: MMATRIX  ( n m size -- )           \ defining word for a 2-d matrix
        CREATE
           OVER , DUP ,
           * * ALLOT
        DOES>  [ 2 CELLS ] LITERAL +
;

\ Structures
\    -----------------------------------
\    | id | m | cell_size |  data area |
\    -----------------------------------

: SMATRIX  ( n m size -- )           \ defining word for a 2-d matrix
        CREATE TYPE-ID ,
           OVER , DUP ,
           * * ALLOT
        DOES>  DUP @ TO TYPE-ID
               [ 3 CELLS ] LITERAL +
;

: MATRIX  ( n m size -- )            \ defining word for a 2-d matrix
     STRUCT-ARRAY? IF   SMATRIX FALSE TO STRUCT-ARRAY?
                   ELSE MMATRIX
                   THEN

;

: }}     ( addr i j -- addr[i][j] )     \ word to fetch 2-D array addresses
               2>R                      \ indices to return stack temporarily
               DUP CELL- CELL- 2@       \ &a[0][0] size m
               R> * R> + *
               +

;

\ Dynamic 2-D array definition,
\    ------------------------------
\    | data_ptr | cell_size | (id) |
\    ------------------------------
```

(fsl-util.seq continues on next page.)

```
: DMATRIX   ( cell_size -- )            \ defining word for a 2-d matrix
        DARRAY
;

: }}fprint ( n m addr -- )        \ print nXm elements of a float 2-D array
        ROT ROT SWAP 0 DO
                        DUP 0 DO
                                 OVER J I   }} F@ F.
                        LOOP

                        CR
                LOOP
        2DROP
;

\ Code for hiding words that the user does not need to access into a hidden wordlist.
\ Private:
\          will add HIDDEN to the search order and make HIDDEN the compilation wordlist.
\          Words defined after this will compile into the HIDDEN vocabulary.
\ Public:
\          will restore the compilation wordlist to what it was before HIDDEN got added,
\          it will leave HIDDEN in the search order if it was already there. Words defined
\          after this will go into whatever the original vocabulary was, but HIDDEN words
\          are accessable for compilation.
\ Reset_Search_Order
\          This will restore the compilation wordlist and search order to what they were
\          before HIDDEN got added.  HIDDEN words will no longer be visible.

\ These three words can be invoked in any order, multiple times, in a
\ file, but Reset_Search_Order should finally be called last in order to
\ restore things back to the way they were before the file got loaded.

\ WARNING: you can probably break this code by setting vocabularies while
\          Public: or Private: are still active.

\ the Vocabulary HIDDEN is already defined in F-PC
\ Vocabulary HIDDEN
variable HIDDEN_SET         HIDDEN_SET Off
variable Private_Used       Private_Used Off
variable OLD_CURRENT        0 OLD_CURRENT !

\ These definitions may require modification for non F-PC systems.
: Public: ( -- )
         HIDDEN_SET @ IF HIDDEN_SET OFF
                         PREVIOUS
                         ALSO HIDDEN
                         OLD_CURRENT @ 0= NOT IF
                                              OLD_CURRENT @ SET-CURRENT
                                         THEN
                      THEN
;
: Private: ( -- )
         HIDDEN_SET @ 0= IF
                           HIDDEN_SET ON
                           GET-CURRENT OLD_CURRENT !
                           Private_Used @ IF PREVIOUS THEN
                           ALSO HIDDEN DEFINITIONS
                           Private_Used On
                         THEN
;

: Reset_Search_Order ( -- )             \ invoke when there will be no more
                                        \ mucking with vocabularies in a file
         HIDDEN_SET @ IF Public: THEN
         PREVIOUS
         Private_Used Off
         0 OLD_CURRENT !
;
```

STRUCTS.SEQ

```
\ structs.fo            An implementation of simple data structures

\ This is an ANS Forth program requiring:
\       1. The words 'Private:', 'Public:' and 'Reset_Search_Order'
\          to control the visibility of internal code.
\       2. The Floating-Point word set
\       3. The compilation of the test code is controlled by the VALUE TEST-CODE? and
\          the conditional compilation words in the Programming-Tools wordset
\ Note that there are two versions of ]] defined, one for ANS the other
\       for F-PC V3.6

\ based heavily upon part of the code described in:
\ Hayes, J.R., 1992; Objects for Small Systems, Embedded Systems Programming,
\ V. 5, No. 3(March) pp. 32 - 45
\ also upon the ideas in:
\ Pountain, D., 1987; Object-Oriented Forth, Implementation of Data
\ Structures, Academic Press, New York, 119 pages,  ISBN 0-12-563570-2
\ and communications with Marcel Hendrix

CR .( STRUCTS            V1.9            3 January 1995  EFC )

Private:

0 VALUE fetch-em            \ execution token of a 'struct-@' (temporary)
V: store-em                 \ a vector to a ',' type word
FALSE VALUE is-const        \ identifies constant or variable type struct
FALSE VALUE GO-EARLY        \ TRUE when doing early binding

: makevar                   \  allocate memory for a struct of given size
   CREATE , ALLOT                       ( size id -- )
   DOES> DUP @ SWAP CELL+               ( -- id addr )
;

: makeconst         \ allocate memory for a constant-type struct of given size
                    \ | id | @ | data... |
   CREATE ,         ( size id -- )
   DROP             \ don't need the size since fetch-em knows it
   fetch-em ,
   store-em         \ lay down the constant structure data
   FALSE TO is-const
   DOES>            ( -- value )
        DUP @ SWAP
        CELL+ DUP CELL+ SWAP @ EXECUTE      \ executes fetch-em
;

: makeinstance ( size --- )           \ create a struct of given size
   is-const IF makeconst
            ELSE makevar
            THEN
;

: ?member-error ( m-id s-id -- )       \ raise an error if s-id and m-id do not match
         OVER OVER

         <> IF      ." Wrong member of structure, STRUCT = " U.
                    ." , MEMBER = " U. CR
               ABORT
            THEN

         2DROP
;

\ calculate address of member base for simple scalar data types
: resolve-scalar-member ( s-id s-addr m-base -- m-addr )
           ROT >R               \ save s-id
           2@ SWAP R>
           ?member-error        \ compare s-id and m-id
           +
;
```

(structs.seq continues on next page.)

```
: resolve-structure-member ( s-id s-addr m-base -- m-id m-addr )
        ROT >R
        DUP 2@ SWAP R>
        ?member-error
        SWAP [ 2 CELLS ] LITERAL + @
        ROT ROT +
;

: resolve-array-member ( s-id s-addr m-base -- m-base m-addr )
          ROT >R
          DUP 2@ SWAP R>
          ?member-error
          ROT +            \ calculate address of array pointer base
;

: aus:          \ Structure member compiler.     | offset | id |
   CREATE OVER , +     ( id offset size -- id offset' )
          OVER ,
   DOES>               \ ( s-id s-addr m-base -- m-addr )
         resolve-scalar-member
;

: smc:          \ Structure member compiler.     | offset | id | struct-id |
   CREATE OVER , +     ( id offset size -- id offset' )
          OVER ,
          TYPE-ID ,
   DOES>               ( s-id s-addr m-base -- m-id m-addr )
        resolve-structure-member
;

Public:

: constant-structure ( '@ ', --  )

           DEFINES store-em
           TO fetch-em
      TRUE TO is-const
;

: structure                              \ Start structure declaration.
   CREATE HERE 0 , 0                     \ ( -- id offset )
   DOES> DUP @ SWAP makeinstance ;       \ ( -- pfa template )

: attribute   ( offset size -- offset' )          \ same as struct:
       >R ALIGNED R>
       STRUCT-ARRAY? IF   smc: FALSE TO STRUCT-ARRAY?
                     ELSE aus: THEN
;

: chars:  ( offset n --- offset' )  \ Create n char member.
   CHARS aus: ;

: char:  ( offset --- offset' )  \ Create 1 char member.
   1 chars: ;

: cells:   ( offset n --- offset' )  \ Create n cell member.
   CELLS  attribute ;

: cell:   ( offset --- offset' )  \ Create 1 cell member.
   1 cells: ;

: struct:  ( offset size --- offset' )  \ Create member of given size.
    attribute ;

: integer: ( offset -- offset' )
     1 cells: ;

: double:  ( offset -- offset' )
     2 cells: ;
```

(structs.seq *continues on next page.*)

```
: float:           ( offset -- offset' )
    FALIGNED 1 FLOATS aus:
;

: endstructure     ( id offset --- )
   SWAP ! ;

\ ===================================================================
\ Words for creating STATICALLY declared arrays WITHIN a structure

Private:

\ For arrays of SCALAR types
: MARRAY:              \     | offset | id | cell_size |
        CREATE                ( id offset n cell_size -- id offset' )
           2 PICK ,  3 PICK ,
           DUP ,
           *
           + CELL+

        DOES>                 ( s-id s-addr m-base -- m-addr )
           resolve-array-member

           \ get cell size and store it in the instance
           SWAP [ 2 CELLS ] LITERAL + @ OVER !
           CELL+
;

\ For arrays of structure types
: SARRAY:              \     | offset | id | t-id | cell_size |
        CREATE                ( id offset n cell_size -- id offset' )
           2 PICK , 3 PICK ,
           TYPE-ID ,
           DUP ,
           *
           + CELL+

        DOES>                 ( s-id s-addr m-base -- m-id m-addr )
           resolve-array-member

           \ get cell size and store it in the instance
           SWAP [ 2 CELLS ] LITERAL + 2@ >R OVER !

           CELL+
           R> SWAP
;

Public:

: ARRAY:       ( id offset size -- id offset' )
       >R ALIGNED R>
       STRUCT-ARRAY? IF SARRAY: FALSE TO STRUCT-ARRAY?
                     ELSE MARRAY: THEN
;

\ ===================================================================
\ Words for creating array pointers WITHIN a structure
\ These ARE NOT dynamic arrays but are general purpose pointers
\ ( does cell_size need to be stored ? )

Private:

: dmpointer:       \ pointer member compiler.     | offset | id | cellsize |
   CREATE OVER ,           ( id offset csize -- id offset' )
          2 PICK ,
          ,           \ store cellsize, but its not being used by anything yet
          CELL+
   DOES>                \ ( s-id s-addr m-base -- m-addr )
         resolve-scalar-member
         @ CELL+
;
```

(structs.seq continues on next page.)

```
: dspointer:       \ pointer member compiler.   | offset | id | struct-id | cs |
   CREATE OVER ,       ( id offset csize -- id offset' )
          2 PICK ,
          TYPE-ID ,
          ,
          CELL+
   DOES>               ( s-id s-addr m-base -- m-id m-addr )
         resolve-structure-member
         @ CELL+
;

Public:

: POINTER: ( id offset cell_size -- id offset' )
       >R ALIGNED R>
       STRUCT-ARRAY? IF DSPOINTER: FALSE TO STRUCT-ARRAY?
                     ELSE DMPOINTER: THEN
;

\ ===================================================================

\ for building arrays of structures and nested structures
: sizeof ( -- n )                \ returns size of a structure, APPLY TO TYPES!!!
    ' >BODY DUP TO TYPE-ID @
    STATE @ IF POSTPONE LITERAL THEN
    TRUE TO STRUCT-ARRAY?
; IMMEDIATE

: typeof ( -- id )               \ returns the type id, APPLY TO TYPES!!!
    ' >BODY
    STATE @ IF POSTPONE LITERAL THEN
; IMMEDIATE

: addrof ( -- addr )             \ return base address, APPLY TO INSTANCES!!!
    ' >BODY @
    STATE @ IF POSTPONE LITERAL THEN
; IMMEDIATE

\ Word to get base address of pointer instance
\ example usage:      pix -> .x{

: ->  ( s-id s-addr -- addr )
      ' >BODY STATE @ IF POSTPONE LITERAL POSTPONE resolve-scalar-member
                      ELSE resolve-scalar-member THEN ; IMMEDIATE

\  usage:   a{ pix -> .x{ ->!
: ->!     ( ar-base addr -- )  SWAP CELL- SWAP ! ;

\ For forcing early binding.
\ These words are written so that they are harmless to invoke at runtime
: [[    STATE @ IF TRUE TO GO-EARLY POSTPONE [
                ELSE FALSE TO GO-EARLY THEN ; IMMEDIATE

\ F-PC V3.6 version
: ]]    GO-EARLY IF POSTPONE ] POSTPONE LITERAL FALSE TO GO-EARLY THEN
; IMMEDIATE

\ ANS version
\ : ]]    GO-EARLY IF ] POSTPONE LITERAL FALSE TO GO-EARLY THEN ; IMMEDIATE

structure STRUCT-HANDLE              \ useful for saving structure instances
       1 CELLS attribute .type
       1 CELLS attribute .addr
endstructure

: h@   ( hdl1 -- hdl2 )  2DUP .type @ ROT ROT .addr @ ;
: h!   ( hdl1 hdl2 -- )  2OVER 2OVER .addr ! DROP ROT DROP .type ! ;

Reset_Search_Order

\ ===================================================================
```

Fast FORTHward

The Prospects for ++Forth

Mike Elola
San Jose, California

If OOLs (object-oriented programming languages) make fashionable a new breed of profoundly extensible programming languages, then OOLs may eventually be a big Forth benefactor.

I have applauded how OOLs promise to make our program designs more open to change. By trivially changing their implementation, we can substantially change their design thanks to the soft-coding of functions and thanks to classes designed for polymorphic use (see my column "Objects Promote New Programming Style," from the preceding issue).

Even if this goal is achieved through other means than extensible data types (object classes), it should become a milestone in the history of computer science.

To conceive of C++ required embracing a bold decision. For years, the evolution of programming languages had tended to hide more and more details of the compiler's manner of operation from the user of the language.

Besides Forth, C++ is one of the few programming languages that bucks the trend. It does this by revealing—rather than hiding—a part of the C++ compiler that is responsible for data types.

...not only data types, but many other substantial extensions can be "plugged in" to Forth.

In the area of data type extension, the C++ user must supply such low-level functions as copy constructors, (object instance) destructors, and so forth. So in a sense, C++ reaches a lower level than C.

Despite the laudable extensibility that C++ achieves, the perilous path it uses to get there may be its undoing. Ironically, the smooth path through which Forth reaches its profound levels of extensibility may never bring it the popularity that it deserves.

Reconcilable C++ and Forth Differences?

If C can be extended into C++, then any conventional programming language can be similarly extended. The programming provisions that C++ includes beyond those of C can serve as a standard basis for achieving object support in a variety of conventional programming languages, including a ++Forth.

However, the nature of the extensibility offered by Forth and by C++ differs considerably.

Forth's compiler extensibility has always given us Forth programmers visibility into a modifiable compiler. For Forth in particular, such visibility and potential for modification extends to all parts of the compiler. That way, not only data types (object classes), but also many other substantial extensions can be "plugged in" to Forth.

Not so long ago, the immaturity of data typing systems escaped notice. C++ is one means to educate ourselves in regard to fully articulated data typing systems. A mature data typing system is very far-reaching. An extensible data typing system has also been shown to be possible, yet complex.

For Forth to support soft-coded functions as do OOLs, it needs an equally far-reaching data typing system. Until that happens, Forth will remain on a less-promising branch in the evolution of programming languages.

Forth can be shown to be amenable to data typing systems, despite its type-vague data stack. In contrast, C++ is not going to be at all amenable to adopting as open, as extensible, or as simple a compiler as the Forth compiler.

While Forth's extensibility reveals a simple compiler, the data type extensibility of C++ reveals an off-putting and complicated compiler: C++ class designers must write the delicate glue routines (constructors, destructors, exception handlers, etc.) that make their data type extensions integrate with one another, and otherwise behave reasonably.

++Forth

If our community wishes to continue to advance our favorite programming language, we may need to prove Forth's ability to adopt data types and objects.

Despite Forth's reputation as a difficult language, the learning curve of C++ is bound to be steeper than Forth's own. The move from C to C++ involves a quantum leap in learning difficulty. Accordingly, if the move from Forth to a ++Forth (with C++ features) involved only a moderate

level of added difficulty, Forth would gain appeal.

C++ has had to pay an awfully high price in clarity for most of its object support. (Nevertheless, the C++ experiment has also proven that older programming languages can be transformed into object-oriented languages.)

Bringing up primitive levels of object support in Forth is simple enough. The worry is that a fully articulated object system may weigh a ++Forth down in much the same way that it weighs down C++.

Beyond the hurdle of adding data typing to Forth, a formal module and object system is needed. Once those hurdles have been cleared, there is much more of C++ to emulate in ++Forth.

In the end, the OOL environment may be irreconcilable with the simplicity of Forth.

Maybe the best we can hope for is that the complexity of C++ will open the minds of the larger programming community to the alternatives that exist. As part of such a scenario, an easy-to-use ++Forth could certainly do us no harm.

++Forth Should Not Overshadow Forth

Even if ++Forth turned out to be substantially less obscure than C++, who will be impressed?

The vastly superior simplicity of Forth has never won it widespread accolades. As paradoxical as it sounds after the previous statement, Forth has developed a reputation as being difficult to learn. So why should we be hopeful that ++Forth would fare any better?

For these reasons, I feel like we need to be able to sell others on Forth first and foremost. If and when it arrives, ++Forth should not be promoted at the expense of normal Forth.

If anything, ++Forth should help entice others on the path towards Forth, which already captures the essence of extensibility and scalability in a programming language. What better proof of this could there be than a ++Forth that can be spun out of Forth naturally and simply?

These sentiments underscore the importance of making Forth more widely recognized as a language that is easy to use and learn. That's where the real challenge lies. One year ago, I proposed some ways to make Forth even easier to use (see "Fine-Tuning Forth, " *FD* XVI/5).

Other than changing Forth in small but friendly ways, Forth books, tutorials, and similar study materials have a very important role to play. To FIG's credit, the re-release of *Thinking Forth* was money that was well spent. Kudos to John Hall for this.

(FORML, from page 18.)

without the interpretation state. The idea behind his model is that we can always compile whatever is read from the input stream, and a special word (`;` in his example) can cause the code which is compiled so far to be executed. This way, control structures like `IF THEN` or `DO LOOP` can be used while "interpreting." (I guess this approach has other advantages, e.g., the need for state-smart words would disappear because `STATE` disappears.)

The free talks during the cheese and wine parties (I am a wine drinker, so California's countless different kinds of wines made a particularly good impression on me) were interesting, especially because of the numerous different topics, not always limited to programming. This was a place where I—a chemist who decided to do automation rather than some more "conservative" branch of chemistry—was not some rare kind of animal to admire, but a perfectly ordinary Forth programmer.

I enjoyed every bit of these talks, especially because in Asilomar I could talk without having to watch constantly to see whether the other person understood what I was talking about. This is something you grow to appreciate when you have to talk "different languages" with chemists and programmers or, even worse, different languages with different chemists and different programmers. As it turned out, many of us started with chemistry or other sciences and did other things than writing programs.

This might explain why Forth is not so popular among mainstream computer scientists and software companies. Forthists seem to be attracted by any challenge and tend to try out very unconventional approaches—definitely a kind of behaviour that dinosaurs like IBM and Microsoft will never tolerate.

The bad part about the FORML conference is that it is so short (about 48 hours altogether). So whatever ideas arose concerning different aspects of the use of Forth—such as integrating Forth into today's operating systems, using Forth as a scripting language, problems with ANS Forth, defining defining words (some very interesting and valuable ideas by John Rible)—could not be discussed in detail because of the lack of time.

The conference ended on the 26th and after lunch we said goodbye to each other and left Asilomar hoping that we will meet next year at the same place. I met nice and fine people in Asilomar, and I was very happy to be able to talk to someone about my work. It was nice to see people face to face who were previously but e-mail addresses. Dr. Ting was kind enough to give me a lift on my way back and showed me around Silicon Valley, giving me a very good impression of the place.

I was probably the youngest attendee at the conference. Not being very young myself, I have to think about what this implies. Are we some endangered species which might disappear in one generation, or is there still a future for Forth?

The Institute for Applied Forth Research, Inc.
Announces the 16th Annual

1996 Rochester Forth Conference on Open Systems

June 19 – 22, 1996
Ryerson Polytechnic University
Toronto, Ontario, Canada

Call for Papers

The 1996 Rochester Forth Conference on Open Systems is hosted by the Institute for Applied Forth Research, Inc. in conjunction with the Southern Ontario Forth Interest Group and McMaster University. The Rochester Conference will again provide a forum for researchers, developers, and vendors to present the latest practical results dealing with open systems. The conference seeks original papers relevant to the design, development, implementation, and use of open systems. Conference topics include:

- Open Firmware Standard/Open Boot™
- Scripting Languages
- Distributed Computing
- Plug and Play™ Systems
- SGML and HTML
- Educational Issues
- Java™

Other areas of interest are Forth programming standards, embedded systems, real-time systems, and the use of Forth for scientific and engineering applications.

Important Dates

February 1, 1996	Deadline for an extended abstract
May 1, 1996	Deadline for the camera-ready copy of final paper

Please send all manuscripts to the Program Chairman.
Additional information will be posted on the world-wide web as it becomes available:
http://maccs.dcss.mcmaster.ca/~ns/96roch.html
Facilities Chair: B.J. Rodriguez

Program Chair
Nicholas Solntseff
Dept. of Computer Science & Systems
McMaster University
Hamilton, Ontario
Canada L8S 4K1
ns@maccs.dcss.mcmaster.ca

Conference Information
Lawrence P.G. Forsley, General Chair
Institute for Applied Forth Research, Inc.
Box 1261
Annandale, Virginia 22003
fax: 703-256-3873 phone: 716-235-0168
lforsley@jwk.com